Automating the Object-Oriented Software Development Process: Workshop Report

Mehmet Aksit and Bedir Tekinerdogan

TRESE project, CTIT, University of Twente
P.O. Box 217, 7500 AE, Enschede, The Netherlands.
email: {aksit | bedir }@cs.utwente.nl, www server: http://wwwtrese.cs.utwente.nl

Abstract. Cost-effective realization of robust, adaptable and reusable software systems demands efficient and effective management of the overall software production process. Current object-oriented methods are not completely formalized and lack the ability of reasoning about the quality of processes and software products (artifacts). There is a need for new modeling formalisms, which enable the quantification of the required quality attributes and support the automation of the object-oriented development process (AOOSD). The ECOOP'98 AOOSD workshop was organized to identify the important issues in this direction.

1. Introduction

Object-oriented technology is increasingly applied by software industry and there is strong evidence that this trend will continue in the near future. To be able to fully utilize the object technology, however, the software companies demand CASE environments that can effectively support the object-oriented software development process. The ECOOP'98 workshop "Automating the Object-Oriented Software Development" (AOOSD) was organized to discuss the important topics in this context. During the workshop, 9 papers were presented which discussed the various perspectives of the automation process. In the following section a framework is presented to classify these papers.

2. The Automation Framework

Figure 1 represents the conceptual framework, which was interactively defined during the workshop to discuss, classify and relate various issues and activities in relation to automating the object-oriented software development process. This framework is structured around three general domains: real-world domain, method domain and computation domain.

As shown by Figure 1, various kinds of persons may be involved in development and usage of a software system. These are symbolically illustrated as multiple stakeholders, end-users, domain engineers, method engineers and software engineers. Some persons may have multiple roles, such as being a stakeholder, user, and a domain expert. The stakeholders have generally different and sometimes conflicting interests. The domain engineer has an in-depth understanding of the application domain. For example, software, which supports developing and applying

S. Demeyer and J. Bosch (Eds.): ECOOP'98 Workshop Reader, LNCS 1543, pp. 474-476, 1998.

insurance products, requires in-depth expertise in this field. The domain engineer may provide this knowledge. The method engineer is responsible to define the method. This can be a general method, such as OMT, or a special method, for example, for developing insurance products. In the latter case, the method engineer has also to be a domain engineer. The software engineer utilizes the method to develop software system.

In Figure 1, the method domain is organized in multiple levels. Each level may be described in terms of a set of artifacts, related knowledge sources, process and management and control activities. An artifact may be defined in terms of its notation, a set of heuristics rules to utilize the artifact, and possibly some constraints. These constraints may be also defined by rules. There are causal relations among the artifacts, since they depend on each other in some order. Sometimes knowledge sources can be expressed within the description of the artifacts, which result in dedicated artifacts. Process may further restrict the causal relations among artifacts. Management and control activities aim at reaching the global objectives of its level. Assume for example that an insurance product has to be developed. At level (i+1), the domain engineer defines some basic insurance concepts. At level (i), by using these concepts, the method engineer constructs a dedicated method for delivering insurance products. At level (i-1), the software engineer applies this method to create an insurance product.

The computation domain enables software systems to be executed on processors. This domain is also characterized by various concerns defined at various levels. Automating the software development process means mapping some of the elements of the real world and the method domains to the elements of the computation domain. High-level mechanisms are defined to ease the mapping process, such as object-oriented languages, agents, dedicated computation models and tools. Through this mapping process, efficiency and effectiveness of the activities in the real-world and method domains have to be improved. Efficiency means that more work can be done for per unit of money. Effectiveness means that the computation process may help the activities to get closer to the objectives.

The papers presented during the workshop can be classified according to this framework. Related to the real-world domain, two papers were presented. The paper “The Case of Cooperative Requirement Writing” by Ambriola and Gervasi aims at supporting cooperation among the stakeholders. The paper “Conceptual Predesign as a Stopover for Mapping Natural Language Requirements Sentences to State Chart patterns”, by Kop and Mayr aims at eliminating the so-called *impedance mismatch* between the real-world and method domains.

The remaining papers mainly correspond to the method layer. Among these, two papers evaluate the state-of-of-the art CASE technology. The paper “Software Quality in the Objectory Process”, by van den Berg evaluates how software quality assurance is realized in Rational Objectory. The paper “Evaluating OO-CASE Tools: OO Research Meets Practice”, by Greefhorst, van Elswijk, Maat and Maijers provides a general evaluation of four object-oriented CASE tools.

Three papers present formalisms to support a set of general-propose artifacts. The paper "Systematic Construction of UML Associations and Aggregations Using cOlOr Framework", by Barbier formalizes relationships among classes. The paper "Formalizing Artifacts of Object-Oriented Analysis and Design Methods", by Saeki defines means to transform artifact specifications to a formal language such as object Z. The paper "Providing Automatic Support for Heuristic Rules of Methods" by Tekinerdogan and Aksit aims at formalizing the heuristic rules of methods such as OMT.

Two papers present tools to help developing dedicated software systems. These environments support various levels in software development and introduce general purpose and dedicated artifacts. The paper "Using the MetaGen Modeling and Development Environment in the FIBOF Esprit Project", by Lesueur, Revault, Sunye and Ziane, presents a meta-CASE tool to develop application specific software systems. This system is tested in a banking domain. The paper "From Visual Specifications to Executable Code", by Enn Tyugu, presents a set of tools which are suitable in producing software systems for simulation and network management.

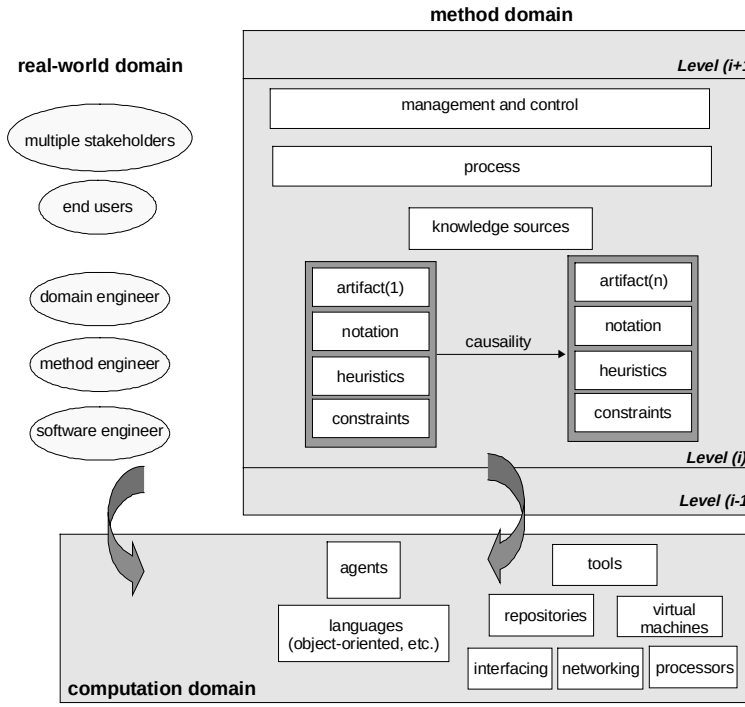

Fig. 1. The context of automation in object-oriented software development.