Softw Syst Model (2003) 2: 73–75 / Digital Object Identifier (DOI) 10.1007/s10270-003-0025-y

Editorial

Model engineering

Published online: 2 July 2003 – © Springer-Verlag 2003

Welcome to the second issue of the Software and System Modeling (SoSyM) journal in 2003.

The heightened awareness of the benefits that can be derived from model-driven software and system development approaches is evident in discussions that take place in academic conferences, workshops, and industry meetings. On the other hand, there is a significant number of developers and researchers who question the feasibility and benefits of model-driven approaches. Advocates of modeling approaches, for sure, can recount the discussions, debates, and outright arguments, they have had with their "code-centric" colleagues. These colleagues recall the dismal failure of CASE tools that promised orders-of-magnitude improvement in productivity and product quality, and the difficulty of maintaining consistency between models and code. The more sophisticated protagonists question the feasibility of developing general-purpose modeling languages and techniques that can provide useful abstractions and mechanisms for most application domains. Two recent events highlight the gap between the two camps. A guest speaker at a recent workshop on Program Comprehension noted that in the COBOL-dominated financial sector "Graphs and UML diagrams may be full of meaning, but unless their meaning can be presented in source terms *(i.e., in COBOL terms)*, they will not be understood by the financial programming community." (editors' comments in italics) – *Comprehending Reality – Practical Barriers to Industrial Adoption of Software Maintenance Automation*, by James Cordy. We also came across a review (on Amazon.com) of an introductory book on Java that included the following complaint: "However, it extensively uses UML, and I think it's a safe assumption that most people who don't know Java, also don't know UML. Looking at UML diagrams can prove to be extremely frustrating for a beginner". Rather than throw our hand up in despair, or scoff at the limited scope of the code-centric view, we should use arguments against the use of models in software development as fuel for driving research on model-driven approaches and development of supporting tools. In this editorial we will sketch our view on some of the research directions that may be worth pursuing.

Model-based software description techniques use models, expressed in a formal language, to describe the architecture of a system, and the behavior of software artifacts. Models expressed in such a language can also be used to (1) clarify the structure of an enterprise (2) describe business workflows, (3) describe development processes and (4) describe the users of software systems in terms of their needs and motivations. These descriptions can be used as communication artifacts, as points of references against which subsequent implementations are verified, or as the basis for further development (e.g., through the use of refinement and realization techniques).

In his work published in the seventies, Stachowiak characterized a "model" as follows:

(1) A model has a purpose.
(2) A model describes some entity that exists or is intended to exist in the future.
(3) A model is an abstraction, that is, it does not describe details of the entity that are not of interest to the audience of the model.

When building a model it is important to consider its purpose. Models can be created to help clarify requirements and designs of complex systems, to support analysis activities, to document systems at various levels of abstraction, and to support decision-making. In a model-driven software development approach, models can also be used to describe how descriptions of software are transformed during development. It is conceivable that a variety of models can be used during different stages of the development process; in general, it would be difficult to develop a model that can be used for all modeling purposes during a development project. To enable software and system developers to use models that best suit their development needs (model purposes), a powerful mechanism for translating between different kinds of models is critical to the successful use of models in practice.

Techniques that describe transformations on software and system artifacts are receiving much attention currently. Transformation models that can be stored, specialized, and reused pave the way for the development of techniques and tools that support systematic and controlled evolution of software and system artifacts. Successful transformation languages can be found in the areas of logic systems, for example, as found in theorem-provers, graph systems in the form of graph transformations, and the database are in the form of schema evolution techniques. Examining the transformation languages in these areas is an ongoing research activity and promises to yield insights that can lead to better approaches to modeling transformations to support evolution of software artifacts. Explicit and possibly standardized model transformation languages will also be key to the development of techniques and methods that support the separation of technology dependent concerns from technology independent concerns during development. This in turn can lead to stable investments in models and transformations.

Use of model-driven development techniques will require adaptation of current software project practices and processes. The use of tools supporting the creation of models and their transformation to efficient executable forms can have a downsizing effect on projects. Using models to generate test cases paves the way for the development of systematic testing processes that utilize software descriptions developed throughout the development phases.

To better support the use of models during development it will become necessary to further understand models in terms of how they can be transformed and analyzed, as well as understand the impact of software size and complexity on model creation, management and evolution. We expect that "model engineering" will become a new and highly interesting sub-discipline in the software engineering discipline.

To summarize:

- Use of models built for specific purposes will become increasingly important for successful development projects.
- Explicit model transformation languages are key to reuse of transformation knowledge and independence of tools and technology.
- Model Engineering might be considered a new subdiscipline to Software Engineering

Summary of papers for this issue

This issue contains four regular papers and an Expert's Voice paper. The Expert's Voice paper from James J. Odell, H. Van Dyke Parunak and Mitchell Fleischer: ***Modeling Agent Organizations using Roles*** deals with the question, how to model large-scale software systems that cross company and may be even social or country boundaries. Based on the notions of agents and roles, they describe how roles interact and how roles can be designed. They identify the collaboration of agents playing specific roles in groups like societies or organizations as the next frontier.

The first regular paper from David Harel and Rami Marelly: ***Specifying and Executing Behavioral Requirements: The Play-In/Play-Out Approach*** describes a promising new approach to requirements elicitation. The method described in the paper is backed by an implementation that allows to iteratively construct complete behaviors from example scenarios that the user is "playing-in". As a result a complete system description is "played-out". This paper is also a successful example of how new theoretic approaches enhance practical software development.

A second regular paper from Marc Frappier and Richard St-Denis is called ***EB3: an Entity-Based Black-Box Specification Method for Information Systems***. In this paper the authors combine the black-box specification method of Cleanroom with the Jackson System Development (JSD) method and the requirements class diagram used in object-oriented analysis. They provide a process algebra and a formal semantics based on input-output relations.

The third regular paper from Florida Estrella, Zsolt Kovacs, Jean-Marie Le Goff, Richard McClatchey, Tony Solomonides and Norbert Toth is called ***Pattern Reification as the Basis for Description-Driven Systems***. The article discusses a pattern-based, object-oriented, description-driven system architecture that is based on pattern and driven by object-oriented models. For that purpose they define an extension to the standard UML four-layer meta-model.

The fourth and final regular paper of this issue from Paris Avgeriou, Symeon Retalis and Nikolaos Papaspyrou: ***Modeling Learning Technology Systems as Business Systems*** demonstrates a n approach to apply an development approach for business systems to the related domain of web-based learning systems. This approach is also a report about development of a reference architecture for learning management systems that will in the future lead to more effective development of this kind of systems and thus might allow to develop tutoring systems as well as advanced deployment technologies in the future.

Further information about SoSyM

A low-traffic mailing group for SoSyM announcements has been created on the internet. Anyone can subscribe, but only the SoSyM Editors are allowed to post to the group (four times a year). If you are interested in receiving SoSyM-related news every issue you may subscribe through our website.

`http://www.sosym.org/`

For further details and updated information on the submission and review process please also see our website.

Sincerely,

Robert France, *Bernhard Rumpe*
Sosym Editors-In-Chief