Softw Syst Model (2013) 12:439–440
DOI 10.1007/s10270-013-0362-4

EDITORIAL

Model-based lifecycle management of software-intensive systems, applications, and services

Robert France · Bernhard Rumpe

Published online: 18 June 2013

The lifecycle of a successful system is the time period that covers all activities associated with developing, configuring, deploying, operating, and retiring the system. Variations in system lifecycles can be expected, for example, differences may arise as a result of the inclusion of physical parts in the system and the number of installations. In addition, software retirement activities may extend over a long period of time, for example, in cases where access to data provided by a system may be required long after the system is terminated.

Lifecycle management has a lot to do with managing the available information about a system. A significant amount of this information can typically be found in the models produced during various development. Software models can thus play a vital role in system lifecycle management. For example, requirement models can be used to support management of requirements, feature models can be used to manage system and user specific variabilities as well as commonalities, and architecture and design models can provide information that support management of deployment and validation activities. The potential role that models can play in lifecycle management raises the following questions: "To what extent do the models produced during software development help (or hinder) lifecycle management?" "Should the software modeling activity be integrated with the lifecycle management of systems, and, if yes, how can this be done?" "What tools are needed to better leverage the use of models in lifecycle management?" "Does a model also have a lifecycle that needs to be managed?"

R. France
Colorado State University, Fort Collins, Colorado, USA

B. Rumpe (✉)
RWTH Aachen University, Aachen, Germany
e-mail: Bernhard.Rumpe@sosym.org

A variety of models may be needed to support lifecycle management, each describing a particular aspect of the system for particular lifecycle management purposes. In such situations, it is important to have an understanding of how the models relate to each other. Such an understanding is needed, for example, to develop appropriate technologies for maintaining consistency across the models and for managing the propagation of changes across the models.

In various conferences, workshops and discussions, we have observed the following model and language integration trends:

- Integration of heterogeneous models and of their corresponding modeling languages remain a challenging research problem. For example, the many semantic variation points in the UML make it difficult to produce an integrated, semantically coherent language in which the variation points are resolved in a consistent manner. It may also be the case that the manner in which the UML notations are integrated vary, leading to the need to support variations on the forms of integration.
- Providing effective support for model evolution is still a pressing problem. Some of the challenging problems include developing support for semantic differencing (diffing) and merging of models. For graphical modeling languages, the lack of support of modularity and encapsulation in modeling languages, as well as their two-dimensional graphical nature, presents challenges; comparing text on a line basis is much easier than comparing model elements arranged in graph structures.
- The need to provide support for tracing information through various models is widely appreciated, particularly in software evolution activities. We suspect that this problem is best tackled by developing language-specific

solutions, rather than general solutions that are language agnostic.

- Language and model integration is particularly challenging when there is a need to model non-software aspects of a system. For example, in the domain of cyberphysical systems, language integration involves providing the means to integrate underlying "communication" paradigms, namely calculi from control theory, physics, and engineering disciplines with the digital state based theory from computer science. Such integration should lead to better model-based lifecycles of these systems.
- We anticipate that a variety of domain specific languages will be more widely used in software development projects, and thus support for integrating DSLs with general purpose modeling and programming languages will be needed.
- Variability of hardware and software is often handled externally to modeling languages, but it may be more effective to provide support for such variability within the languages. An approach to managing variability in product lines that is built on top of modeling languages may help to some extent, but a language-integrated approach that leverages context conditions and language semantics may be more effective.
- Semantic integration of models is furthermore needed in situations in which integrated models are used as the basis for static or dynamic analysis (e.g., formal analysis of functional properties and simulation). Few integration techniques adequately address semantic issues.
- Recent research has focused on the use of models at runtime to support runtime adaptation of software. For example, in plant control systems an explicit representation of the controlled plant that faithfully captures monitored aspects of the plant can be used as the basis for adapting the plant control software. Runtime models cannot be distinguished anymore from requirements and design models produced during development, in terms of the abstractions they embody. From this perspective, the lifecycle of requirements and design models extends beyond the development and maintenance phase into the operational phases. In fact, using these models at runtime makes the models an integral part of the operation of the system, while at the same time, enables evolution of the running system through evolution of the runtime models. This integrated co-evolution can be viewed as a form of lifecycle management based on these models.

In summary, the use of models in system lifecycle management raises interesting and challenging research opportunities. Furthermore, we in the software and system modeling community cannot ignore lifecycle management issues: As the use of models becomes more widespread, the need for lifecycle management of models will become necessary. Sound lifecycle management of development artifacts is a core competence of integrated software-intensive systems development and becomes even more pressing in the context of globalized software development environments.