Acta Univ. Sapientiae, Informatica, **1**, 1 (2009) 45–52

The use of development models for improvement of software maintenance

Ján Kunštár
Dept. of computer and informatics
Technical university of Košice,
Letná 9, 042 00 Košice, Slovakia
email: jan.kunstar@tuke.sk

Iveta Adamuščínová
Dept. of computer and informatics
Technical university of Košice,
Letná 9, 042 00 Košice, Slovakia
email: iveta.adamuscinova@tuke.sk

Zdeněk Havlice
Dept. of computer and informatics
Technical university of Košice,
Letná 9, 042 00 Košice, Slovakia
email: zdenek.havlice@tuke.sk

Abstract. Nowadays, the cost of software system is one of the most important factors for choice of certain system by customer. Recent trends in software and system development have revealed the asset of usage of the abstract models through the software life cycle's phases. Abstract models streamline and speed up not only development but suitable models can also improve maintenance process to be more effective and safe. Presented paper briefly analyses SysML, which supports development process of complex systems. Main part is oriented to new approach to model driven system development supporting SysML concept named System Development Unified Process (SDUP) extended by concept of Model-Driven Maintenance.

1 Introduction

These days, models present one of the most important considerations of system and software development. Model-based design supports exploratory design

AMS 2000 subject classifications: 68N99
CR Categories and Descriptors: K.6.3 [Software Management]: Software maintenance
Key words and phrases: software maintenance, software system life cycle, system modeling language (SysML)

and analysis by allowing designers to effectively represent and investigate their knowledge about the system during the decomposition and definition process. Additionally, experiments can be performed on models to eliminate poor design alternatives and to ensure that a preferred alternative meets stakeholder objectives. This modeling concept stays at the core of Model Driven Architecture (MDA). However, one of the most important factors of modeling, in order to support the MDA development process, is the choice of modeling language. To support model-based design and to overcome some limitations related to Unified Modeling Language (UML) strict software focus, the Object Management Group (OMG) has developed the Systems Modeling Language (SysML) [1].
In this paper, a methodology for model-based system development using the SysML is presented with emphasis on Model-Driven Maintenance (MDM) which utilizes development models for improving software maintenance.

2 Model-driven system development and modeling languages

Model-driven architecture [2], defined and supported by the OMG, defines an approach to IT system specifications that separates the system functionalities from the implementation details on a particular technological platform. The MDA [1] is a framework for model driven software development defined by the OMG which has elevated the software development to the next step. Using MDA, it is possible to have an architecture that will be language, vendor and middleware neutral.
One of the key standards that make up the MDA is the UML [1]. UML has proved immensely popular with software engineers, but its software focus has discouraged many system engineers from adopting it earnest. The OMG customization of UML for systems engineering in form of new modeling language called SysML is intended to support modeling of a broad range of systems, which may include hardware, software, data, personnel, procedures, and facilities.

2.1 SysML

OMG SysML is a visual modeling language for systems engineering that extends UML 2 in order to analyze, specify, design and verify complex systems, intended to enhance systems quality, improve the ability to exchange systems

engineering information amongst tools and help bridge the semantic gap between systems, software and other engineering disciplines [1]. OMG SysML reuses a subset of UML 2 concepts and diagrams and augments them with some new diagrams and constructs appropriate for systems modeling. The benefits of using SysML in system development process are following [1], [3]:

- SysML semantics are better suited for systems engineering. SysML reduces UML software-centric restrictions and adds two new diagram types for requirements engineering and performance analysis.

- SysML allocation tables support various kinds of allocations. These tables support requirement, functional and structural allocation, thereby facilitating automated verification and validation and gap analysis.

- SysML's requirement modeling support provides the ability to assess the impact of changing requirements to a system's architecture.

- SysML is a precise language, including support for constraints and parametric analysis which allows models to be analyzed and simulated.

- SysML is an open standard and supports XMI and ISO 10303-303 (AP233) allowing for information interchange to other systems engineering tools.

OMG SysML includes diagrams that can be used to specify system requirements, behavior, structure and parametric relationships. These are known as the four pillars of OMG SysML [1]:
I. Structure. The block is the basic unit of structure in SysML and can be used to represent hardware, software, facilities, personnel, or any other system element. The system structure is represented by block definition and internal block diagrams.
II. Behavior. The behavior diagrams include the use case diagram, activity diagram, sequence diagram, and state machine diagram. The extensions made to standard UML activity diagrams support the compatibility with widely used EFFBD notation that will facilitate and improve interaction between SysML and traditional software engineering tools and facilitate the migration to SysML [3].
III. Requirements. The requirement diagram is a new SysML diagram type that captures requirements hierarchies and the derivation, satisfaction, verification and refinement relationships. This diagram provides a bridge between typical requirements management tools and the system models [3]. Hence requirements become an integral part of the product architecture [1].

IV. Parametrics. The parametric diagram is a new SysML diagram type that describes the constraints among the system's properties associated with blocks. This diagram is used to integrate behavior and structure models with engineering analysis models such as performance, reliability, and mass property models.

3 System development unified process

Presented model of system development life cycle includes all phases typical for the most of common life cycle models. However, within this model, the modifications regarding the MDA development approach using SysML were required. The model emphasizes the maintenance phase and its impact on the whole system development process (Section 4).

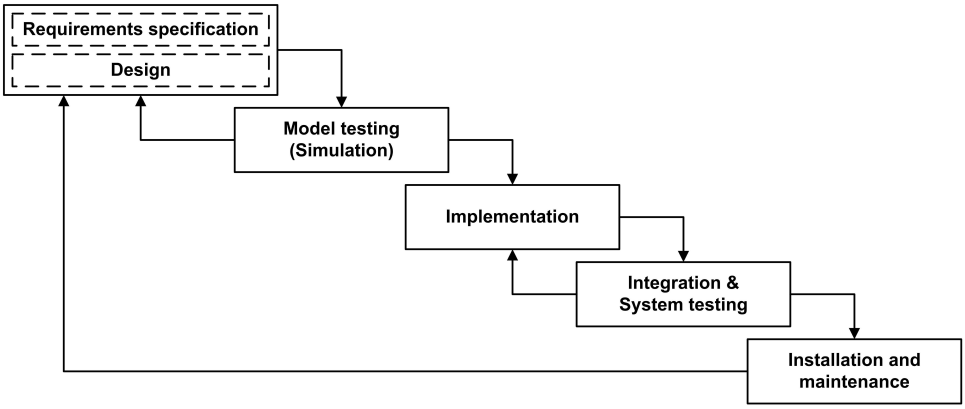

Figure 1: System development unified process

In general, the presented model (Figure 1) may be considered as having five distinct phases, described below:
1. Integrated phase that includes phases of requirements specification and design of the system. By means of using SysML as modeling language, it is possible to integrate these two previously distinct phases into one using the parametric, requirement and design models [1], [3]. This step involves gathering and defining the system's requirements that are directly related to design models with a high level of abstraction that is independent of any implementation technology (platform independent models).
2. Model testing. This phase consists of using the models created in pre-

vious step to be methodically verified to ensure that they are error-free and fully meet the specified requirements. This testing can be processed in form of simulation using the properties of SysML parametrics.
3. Implementation. In this step, the platform independent models are transformed into system's platform specific models that are linked to specific technological platforms (e.g. programming language, operating system or database) [2], [4]. These models are afterwards transformed into implementation artifacts as executable code and database schemas.
4. Integration and system testing. In this stage, both individual system components and the integrated whole are methodically tested and evaluated regarding to technological platforms and quality and reliability of system's performance.
5. Installation and Maintenance. This step involves preparing the system for installation and use at the customer site. A maintenance part involves making modifications to the system or individual component to alter attributes or improve performance. These modifications arise either due to change of requirements, or defects uncovered during system's testing. The main difference compared to standard system maintenance is that no change in system can be processed without accordant modification in design/requirement models (Section 4).

4 Model-driven maintenance

Program comprehension, impact analysis and regression testing are the most challenging problems of software maintenance in the present [5]. An inconsistent state of the software artifacts markedly contributes to all three mentioned problems. Each software system consists of artifacts (e.g. source code, documentation, makefile, models of system) which describe only a limited part of the software and the actual system is their composite. If all system artifacts are't in consistent state, they can't be used together as the source of knowledge about the system. This rapidly decreases the ease with which a software system or its component can be modified during its operation – the maintainability of software system.

4.1 Model-driven maintenance process

Model-driven maintenance process is one useful aspect of knowledge-based software life cycle oriented to better usability of all analysis, design and implementation models in maintenance of systems [6]. MDM is based on uti-

lization of knowledge from the system models and dependences among them for improving maintenance process. Inspiration for MDM is the MDA. MDA concentrates on development of software system using UML as programming language. The direction of progress is from models to application's code. If the change of the system needs to be done according the consistency rules of SDUP (Section 3), it is important to come back to system models, so reverse engineering needs to be used.
In MDM, models of system are the basis for whole maintenance process and therefore there is a requirement to preserve essential models together with the code of application. These essentials models are taken from project database and joined to conjunctive preservation during the installation phase.
Knowledge from essential models, which is element of application, allows us to go cyclically through the phases of life cycle during the maintenance process without the need of browsing project database.

4.2 Model-driven maintenance life cycle

The main difference between the life cycle of normal software maintenance and MDM is in the phase of software system life cycle where the maintenance starts. Normal maintenance life cycle starts with the operation of software system. As system is used, requirements for error correction or requirements (user defined or as a consequence of environment change) for change of the system are detected. The last phase of maintenance life cycle is modification of the system itself. After modification, the system returns to the operation again.
The view that maintenance is strictly a post-delivery activity is one of the reasons that make maintenance hard. According to Pigoski's definition of software maintenance [7], it is very important to prepare software system for its modifications still during the development of the system and not only after delivery. Therefore MDM starts as early as during the system development by conjunction of essential models to application's code. MDM life cycle has the same phases like mentioned normal maintenance life cycle. The difference is in the way how the changes are performed at the basis of the user's requirements. On Figure 2 is displayed the modification process in accord with MDM.
As a first step, the requirements need to be well specified because it is very important to avoid the misunderstandings between users of the system and maintenance programmer. In here, active user participation is very important. Unfortunately most users don't understand the complex diagrams preferred by many traditional modelers. Solution presents the adoption of inclusive models

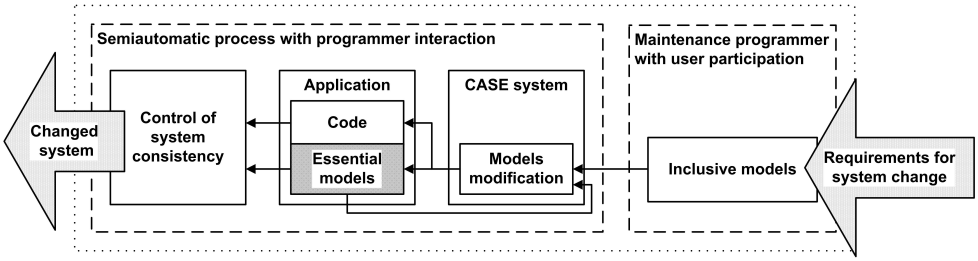

Figure 2: Modification of software system during MDM

which are used to help capture and analyze requirements for certain system [8]. The maintainer can build the requirements diagram after all requirements were exactly specified by users.
After requirements are well specified, maintainer can modify essential models which are part of application. The use of extern CASE system for visualization and applying of changes is useful in this phase. The changes of models can be done without modification of working software system. In this way, the maintenance programmer is able to discover impacts of required changes before they are really implemented to the code of system. When a programmer knows about all required changes he can implement them all in one step, without impacts to unchanged part of the system.
After all, when required modifications are implemented to the code and to the models of application, the consistency control needs to be done. If all changes done in models were processed also in the code, they both describe the same system - they are in consistent state.

5 Conclusions and future work

This paper presents the SDUP, which support the concepts of MDM. This approach based on the conjunctive preservation of program code and models, supports consistency between the essential models and code, as no change in code can be processed without accordant modification in system's models. MDM utilizes knowledge acquired from system's abstract models for uncovering the unwanted side effects of required changes before they are really performed to the system's code. Utilization of system's models streamlines maintenance process and also helps to system comprehension.
In our next research we want to complete realization of proposed MDM. We will work on the proper format of knowledges acquired from essential models.

We will also perform an experimental confirmation of contribution of proposed approach to software maintenance.

Acknowledgement

This work was supported by VEGA Grant No. 1/0350/08 Knowledge-Based Software Life Cycle and Architectures.

References

[1] OMG Systems Modeling Language (OMG SysML) v 1.0 (07-09-01), *OMG Available Specification*, http://www.omgsysml.org/.

[2] A. Kleppe, J. Warmer, W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, 2003, 192 pp, Addison Wesley, ISBN 0-321-19442-X (2003).

[3] T. Weilkiens, *Systems engineering with SysML/UML: modeling, analysis, design*, 322 pp, Morgan Kaufmann Publishers, ISBN: 978-0-12-374274-2 (2007).

[4] A. Benczúr, Z. Hernáth, Z. Porkoláb, LORD: Lay-Out Relationship and Domain Definition Language, *10th Advances in Databases and Information Systems*, Ed: Yannis Manolopoulos et al., Thessaloniki, pp. 215-230 (2006).

[5] G. Canfora, A. Cimitile, Software Maintenance, *Handbook of Software Engineering and Knowledge Engineering*, volume 1. World Scientific, 2001, ISBN: 981-02-4973-X (2001).

[6] Z. Havlice et al., Knowledge-based software life cycle and architectures, *Computer Science and Technology Research Survey*, Košice, ISBN 978-80-8086-071-4 (2007).

[7] T. M. Pigoski, *Practical Software Maintenance Best Practices for Managing Your Software Investment*, John Wiley & Sons, New York, (1997).

[8] S. W. Ambler, R. Jeffries, *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*, John Wiley & Sons, New York (2002).

Received: October 13, 2008