doi: 10.1515/acss-2014-0010
2014/16

Applied Computer Systems

Implementation of Software Configuration Management Process by Models: Practical Experiments and Learned Lessons

Arturs Bartusevics[1], Leonids Novickis[2], Stefan Leye[3]
[1–2] *Riga Technical University, Latvia*, [3] *Fraunhofer Institute for Factory Operation and Automation IFF, Germany*

***Abstract* – Nowadays software configuration management process is not only dilemma which system should be used for version control or how to merge changes from one source code branch to other. There are multiple tasks such as version control, build management, deploy management, status accounting, bug tracking and many others that should be solved to support full configuration management process according to most popular quality standards. The main scope of the mentioned process is to include only valid and tested software items to final version of product and prepare a new version as soon as possible. To implement different tasks of software configuration management process, a set of different tools, scripts and utilities should be used. The current paper provides a new model-based approach to implementation of configuration management. Using different models, a new approach helps to organize existing solutions and develop new ones by a parameterized way, thus increasing reuse of solutions. The study provides a general description of new model-based conception and definitions of all models needed to implement a new approach. The second part of the paper contains an overview of criteria, practical experiments and lessons learned from using new models in software configuration management. Finally, further works are defined based on results of practical experiments and lessons learned.**

***Keywords* – Model-Driven Approach, Software Configuration Management.**

I. Introduction

Nowadays software configuration management is not only a challenge to choose a correct source code management strategy for particular projects [1], [8], [10]. During the last years, software development projects have become bigger and complex in context of source code lines, components, developed using different technologies, and dependencies from other software. At the same time, new versions of a developed product should be released as soon as possible without failed builds, unexpected errors or invalid set of configuration items [10], [1], [2]. To achieve this, new solutions to software configuration management try to combine several tasks, such as source code management, continuous integration, build management, release management, bug tracking etc. Usually software development companies already have a set of tools to solve the above-mentioned tasks. The main challenge is to increase reusability of developed solutions and configurations of tools to reduce efforts for manual steps and for implementation of the same solutions in new projects.

A. Problem Formulation

The following problems are discussed and underlined in the current paper:

- Lack of connections between general software configuration management process and implementation of particular parts of process by different tools. This increases efforts for configuration and customization of particular tools and many manual steps required to support an overall process completely.
- Solutions to particular tasks of configuration management are very specific for concrete project and are not reusable. The main reasons of this problem are mixtures of different parts of configuration management in one solution. For example, build script contains hardcodes for a specific project, absolute paths of server directories, elements of source code management, connections to a bug tracking system etc. There are many efforts required to adopt such a script for other project. There is a lack of methodologies on how to design parameterized, reusable solutions to particular tasks of configuration management that is not dependent on other tasks.

B. Scientific Novelty

The paper provides a new model-driven approach to the implementation of software configuration management process. Unlike other approaches and solutions, a new approach is not oriented to a particular tool that “should solve any problem” but provides the steps how to increase the reuse of existing solutions with well-known tools for source code management, continuous integration, bug tracking and build management. New approach contains three levels of models to describe a configuration management process in context of different abstraction levels. The models provide an approach on how to design reusable and independent solutions and the way from the general process overview to concrete technical solutions. There are five practical experiments described in this paper to show gains and benefits of a new approach. As a result, a number of lessons learned are provided that could help to improve a new approach in future and could generate new ideas.

C. Paper Structure

Section II of this paper provides an overview of related research in software configuration management area to detect trends of improvements, lack of existing approaches and to better underline the novelty of the current paper. In Section III, description of a new approach is given with short explanations and definitions of models. Next sections provide the description of criteria to evaluate gains of a new approach, results of experiments and practical lessons learned during experiments. Finally, conclusions are given, and directions of further work are defined.

II. Related Work

As far back as 1992 there was published an article [9] that introduced to main challenges of configuration management area. One of the main ideas is related to the development of a configuration management service model. Many things have changed since then; more standards have been developed in software development field, new tools for configuration management designed. In a recent interview with a long-term expert in configuration management area [10], the year 1998was mentioned, when there was an attempt to create a "super tool" for integration of all solutions of configuration management in one framework. Attempt failed, because solutions were too complicated. Configuration managers and developers were afraid of "majesty" of such a tool. Configuration management expert [10] emphasizes a challenge to enhance trust between configuration managers and developers as the main future challenge. The main requirement for this is a clear procedure, which could be trusted by programmers. Other configuration management experts [1], [2] note that solutions will be ineffective and will require additional resources without planning of general process before implementation of particular solutions and installation of tools. Modern solutions require reusable approaches that allow coming efficiently from the process general requirements to technical implementation.

Analyzing different approaches of reuse oriented solutions, more ideas from MDA [6] have been found. The important task in configuration management is source code management, and significant part of model-driven solutions is related to this task [11], [12], [13]. New approaches try to improve source code management by modeling product components, streamlines and branches [13]. Abstract models are designed to improve new development of source code management systems [11], [12]. There are solutions that provide an abstract model for general configuration management process based on software quality standards [14], [15], [16]. Usually the approaches do not provide how to increase reuse of existing solutions. It can be very important because software development companies usually have a set of concrete tools that are trusted from their point of view. Thus, new tools or methods with "super performance", "mystic full-automated level" cannot be trusted and acceptable by companies.

The following studies [7], [3], [4] consider a software configuration management process as a whole, not just a particular task. Solution in the article [7] has been designed as a general concept of configuration management and meta-model for creating different models of software configuration. The solution is focused on projects, where development is based on a model-driven approach, but there are no explanations on how this approach could be used in projects with other development approaches.

The main concept of configuration management in study [4] was taken from the ITIL (Information Technology Infrastructure Library) standards and later an abstract model was designed. Later this model could be transformed to a platform specific model. Although this solution also includes the implementation for model-driven configuration management, it is focused on a single technology (JAVA). No recommendations are provided on how to integrate together different tasks of configuration management, such as source code management, bug tracking, build management.

Study [3] focuses on various ways of mutual integration of configuration management different tools. To maintain a full configuration management process, a number of tools are required: version control systems, bug tracking systems, build servers, continuous integration servers etc. The practical experience indicates that all tools often work separately from each other. The main scope of solution is to integrate different tools to solve all tasks for configuration management. To integrate various configuration management tools together, the definition of general concept of each integrated tool is required [3]. The paper offers an ontology for a configuration management process. This ontology provides a general configuration management model and shows how various configuration management tools should be integrated. The study does not have any guides how the ontology could be used for a particular project. It is not clear what kind of ontology editors are advised to use and how to determine the moment when the changes have to be made.

The new model-driven approach provided in the current paper supports the main idea of described related studies about models. Unlike related studies, models in the provided solution have strong defined connections between each other and provide a full way from an abstract process overview to concrete implementation of particular tools, scripts or frameworks. This could reduce efforts for invalid customization of technical solutions. Additionally, a new approach is not oriented to any specific tool but allows using existing, well-known and trusted tools. The approach provides only a methodology on how to refactor the existing solutions and design a new one in order to increase its reuse. This could save up time for implementation of similar solutions for other projects.

III. Model-driven Approach to Software Configuration Management

Development of a new model-driven approach to configuration management was started with the position that different processes of software development required a set of instances or environments. It means that software product could not be developed, tested and used in one environment. Usually different environments are used for the mentioned actions, for example, DEV environment for development, TEST environment for testing and PROD environment for real-time exploitation of software product. From the perspective of configuration management, the main scope is to move changes in a product from one environment to another at a particular time moment. According to development methodology, parallel developments in particular projects, software product lines and some other factors, different sets of environments and different flows of changes could be used for projects. Thus, firstly, environments and flows of changes should be modeled to describe a general software configuration management process. Secondly, after all environments and all flows are known, all actions should be defined that required to implement each flow of changes. For, example, the following actions are required to move software changes from DEV to TEST environment: prepare baseline of a source code, build executable from a source code, deploy executable to TEST environment. Finally, particular solutions should be selected for each mentioned action. New approach requires designing solutions structured by actions. For example, a company may have a few different solutions (scripts, function, framework etc.) to build executable from a source code. Any solution should be parameterized and independent of solutions of other actions. For example, build script should receive a set of parameters and return executable. Script should not contain any specific hardcodes or any information about actions of source code management, bug tracking, servers where executable should be deployed etc.

There are three levels of models in the provided approach:

- Environment Model (EM) – provides a model of all instances included in software development project. A model also contains all flows of software changes between different environments. This model provides an overview of general infrastructure of project in context of instances. Based on state of environments and flows of changes, general risks of configuration management could be detected.
- Platform Independent Action Model (PIAM) – provides a set of actions needed to apply all flows from the Environment Model. The actions are abstract and do not contain any details specific for a particular platform. For example, action “Compile” should be used to compile software from a source code but in this model any details about a software technology, compilation algorithm, and platform are not known.
- Platform Specific Action Model (PSAM) – provides an extended variant of Platform Independent Action Model, because the same actions are fulfilled with details about a platform, technology, specific scripts etc. In this model, action “Compile” already has information on a technology, compilation algorithms, platform etc. It means that in this model all details are known, for example, it could be ANT build script for JAVA projects.

General picture of a new model-driven approach is provided in Fig. 1.

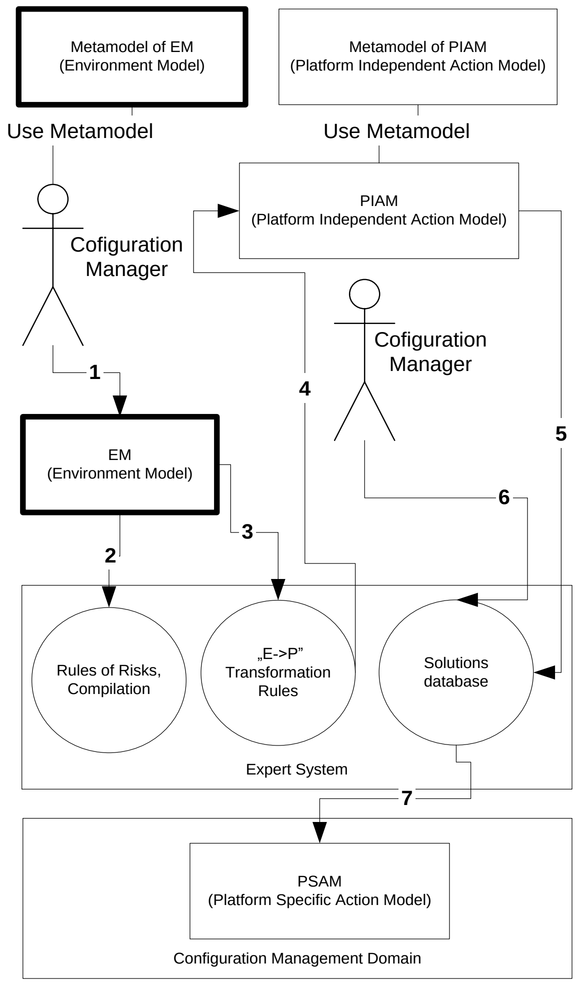
Metamodel of EM
(Environment Model)
Metamodel of PIAM
(Platform Independent Action Model)
Use Metamodel
Use Metamodel
Cofiguration Manager
PIAM
(Platform Independent Action Model)
Cofiguration Manager
1
4
5
EM
(Environment Model)
6
3
2
Rules of Risks, Compilation
„E->P” Transformation Rules
Solutions database
Expert System
7
PSAM
(Platform Specific Action Model)
Configuration Management Domain

Fig. 1. Model-based approach for software configuration management.

Illustration of a model-driven approach in Fig. 1 is represented as a flow with interactions from a configuration manager. Arrows with numbers mean particular stages of the approach. The first stage “1” provides creation of Environment Model from a special meta-model. Configuration manager builds the Environment Model from a set of components from the mentioned meta-model. During the second stage “2”, the created Environment Model should be compiled by a special block in the Expert System, called “Rules of Risks, Compilation”. Expert System in context of this research is a special warehouse for different blocks of rules and a database with ready solutions of actions. After stage “2” a configuration manager compiles the Environment Model with a description of general configuration management risks if such exist. Stage “3” explores the ready Environment Model by a special block of the Expert System called “E->P”. The main task is to detect actions needed to apply each flow between environments. During stage “4”, the

Platform Independent Action Model performs using actions defined at stage "3" and meta-model of PIAM. The stages "5" and "6" require the second interaction from a configuration manager to analyze the ready Platform Independent Action Model and to choose solutions for each action from "Solution Database". Structure of "Solution Database" is provided in Fig. 2.

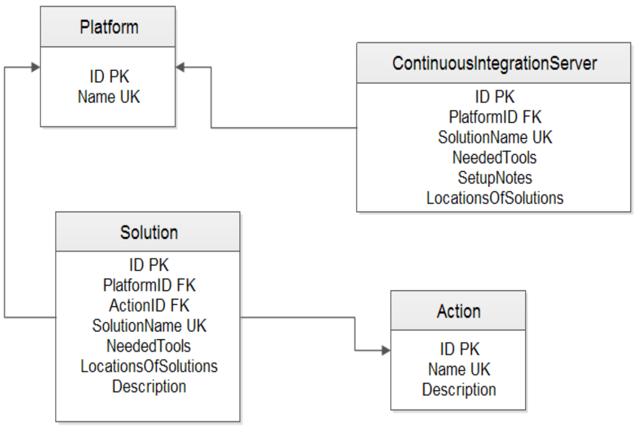

Fig. 2. Structure of Solution Database.

Solution Database contains all information about all configuration management actions described in the PIAM model. For example, action "Compile" could have five different solutions to compile software from a source code for the following technologies: JAVA, Ruby, C++, Oracle, C#. The mandatory requirement is that all solutions are parameterized and do not have dependencies on solutions of other actions. For example, a compilation script should not know any details about other actions from PIAM, hardcodes from bug tracking management, hardcoded hosts, absolute paths etc. All necessary things should be given as parameters. Any solution stored in the Solution Database has the following attributes:

- ID – a unique identifier in the database;
- PlatformID – a reference to a platform;
- ActionID – a reference to an action. Table "Action" contains all possible actions from the PIAM meta-model;
- NeededTools – a set of tools to implement this solution;
- LocationsOfSolutions – information about ready scripts, frameworks, functions, including paths, locations of servers, web-pages etc.;
- Description – some notes provide additional information about implementation, specific technical details.

During the last stage "7", work with the ready PSAM model is required. Each solution of each action should be technically applied in the Configuration Management Domain according to information in the Solution Database.

IV. Criteria of Evaluation of Model-driven Configuration Management

To calculate criteria of the described approach, the following parameters should be fixed in projects, where the mentioned approach should be implemented:

- **AVG_H_OLD** – Average count of hours per week needed to support a configuration management process before implementation of a new approach;
- **AVG_H_NEW** – Average count of hours per week needed to support configuration management after implementation of a new approach;
- **IMPL_TIME** – Count of hours used to implement a new model-driven approach;
- **REM_TIME** – Count of weeks till the end of a particular project (contract end date);
- **FAILED_BUILDS_BEFORE** – Count of failed builds of product during last two months before implementation of a new approach;
- **FAILED_BUILDS_AFTER** – Count of failed builds of product during last two months after implementation of a new approach;
- **GENERAL_BUILD_COUNT_BEFORE** – General count of builds during last two months before implementation of a new approach;
- **GENERAL_BUILD_COUNT_AFTER** – General count of builds during last two months after implementation of a new approach;

The mentioned parameters are oriented to evaluate time spending for configuration management processes in different projects before and after implementation of a new approach. Parameters related to time are exported from a time management system in a software development company, where experiments have been implemented. In the mentioned time management system, each developer and configuration management should fix time that he spent for a current project. This procedure allows extracting the above-mentioned time parameters (AVG_H_OLD, AVG_H_NEW, IMPL_TIME) automatically. A parameter REM_TIME was taken from the contract of a particular project, but other parameters were exported from a build management system, where any build in a particular project is fixed.

Thanks to the above-mentioned parameters, the following criteria for a model-driven approach are defined:

Short Time Gain – percent that shows changes in time needed for configuration management before and after implementation of a new approach. Criteria should be calculated by (1):

$$Short\_Time\_Gain = \left(1 - \frac{AVG\_H\_NEW}{AVG\_H\_OLD}\right)*100 \quad (1)$$

Long Time Gain – percent that shows changes in time needed for configuration management before and after implementation of a new approach, but taking in account time till the end of project. For example, if implementation of a new methodology needs one month, but with an old solution only half a month is needed till the end date of project, a Long Time Gain will be negative. However, Short Time Gain could be positive and quite optimistic. Some projects could spend 2 hours per week for configuration management before implementation of a new approach and one hour after. It means that Short Time Gain will be 50%. At the same, only some months could remain till the end of project, but implementation of a new approach could take some weeks. In this case, implementation of a new approach is not efficient. Long Time Gain criteria could be calculated by (2):

$$Long\_Time\_Gain = (1-(IMPL\_TIME + \\ + \frac{AVG\_H\_NEW * REM\_TIME}{AVG\_H\_OLD * REM\_TIME}))*100 \; . \qquad (2)$$

Failed Build Difference – percent that shows the difference of failed builds before and after implementation of a new approach in a particular project. The criteria could be calculated by (3):

$$FAILED\_BUILD\_DIFF = \\ = (1-\frac{FAILED\_BUILDS\_AFTER}{FAILED\_BUILDS\_BEFORE})*100 \qquad (3)$$

General Build Difference – percent that shows the difference of general count of builds before and after implementation of a new approach in a particular project. The criteria could be calculated by (4):

$$GENERAL\_BUILD\_DIFF = \\ = (1-\frac{\text{GENERAL\_BUILD\_COUNT\_AFTER}}{\text{GENERAL\_BUILD\_COUNT\_BEFORE}})*100 \qquad (4)$$

V. Overview of Experiments

New model-driven approach to the implementation of software configuration management has been implemented in five different projects during the last 4 months. The owner of all projects is one of the biggest IT companies in Latvia. The projects are related to the implementation and support of different software tools for different enterprises in Latvia. During experiments, the following set of technologies and tools have been used for software development and technical implementation of software configuration management processes: Oracle, JAVA, Ruby on Rails, Subversion, Git, JIRA, Jenkins, Bamboo, and Hudson. The following steps have been done in each of the mentioned five projects:

- General presentation of a new model-driven approach to all developers and configuration managers in a particular project;
- Creation of an Environment Model;
- Transformation of an Environment Model to a Platform Independent Action Model, discussions with developers and managers to validate results of transformation;
- Refactoring of existing solutions and saving information about refactoring solutions in the Solution Database;
- Creation of Platform Specific Action Model by selecting solutions for particular actions from the database mentioned before;
- Implementation of a Platform Specific Action Model;
- Fixing and calculating parameters and criteria described in previous section of this paper.

Table I provides an overview of parameters and calculated criteria after the experiments described above.

TABLE I
RESULTS OF EXPERIMENTS

	Parameters								**Criteria**			
Projects	**AVG_H_OLD**	**AVG_H_NEW**	**IMPL_TIME**	**REM_TIME**	**FAILED_BUILDS_BEFORE**	**FAILED_BUILDS_AFTER**	**GENERAL_BUILD_COUNT_BEFORE**	**GENERAL_BUILD_COUNT_AFTER**	**Short Time Gain (%)**	**Long Time Gain (%)**	**Failed Build Difference (%)**	**General Build Difference (%)**
1	25	2.4	160	52	15	4	84	87	90	78	73	−4
2	40	1.5	700	36	5	1	160	92	96	48	80	43
3	2	1	200	104	3	2	65	67	50	−46	33	−3
4	15	2	50	52	10	3	73	71	87	80	70	3
5	12	2.5	320	52	7	5	88	91	79	28	29	−3

A. Main Conclusions of Experiments

In conclusion, all parameters and criteria have been taken from Table I.

The analysis of experiments started with criterion "General Build Difference". This criterion can answer the question "Is criterion about gain objective". It means that an increase in general count of builds automatically requires more time resources to make these builds. Thus, for example, if new count of builds is greater than that before experiments, time gain becomes more significant. Results of experiments in project "2" show that short time gain is 96 %, but number of builds after experiments is greater by 43 %. It means that the refactoring of existing solutions for configuration management in project "2" improved a general process very well. In other projects – "1", "3", "4" and "5", the difference in general count of builds is relatively low: 3 % or 4 %. In these projects, a new approach saves time resources only, but does not change all process globally.

Actually, a long time gain can better show how successful experiments are. If difference between short and long time gains is low, it means that results of experiments could be more believable. According to this suggestion, projects "1" and "4" are more successful than other projects. The gains in the above-mentioned "1" and "4" projects are near 80%, it means that the implementation of a new model-driven approach can save more resources. However, long time gain in project "3" is negative, it means that implementation of a new methodology is not efficient. During implementation of a new approach for project "3", much more time (200 hours) has been spent for refactoring of existing solutions, but short time gain is only 50 %. After this experiment, a careful analysis has been performed to detect reasons why refactoring requires so much time. The main reason is a version control system. In projects "1", "2", "4" and "5", a Subversion system has been used to control changes in a source code, but in project "3" it has been another system – Git. This fact requires much more time for the refactoring of solutions related to version control because of lack of experience using Git.

VI. LESSONS LEARNED

In this section most important lessons are described that have been learned from mentioned experiments. These lessons could define further research to improve a model-based approach to configuration management.

A. Lesson 1

Implementation time of a new model-driven approach to configuration management should be decreased. It could be achieved by creating services for integration of different couples of tools. For example, a full-automated solution to release management that is part of configuration management from a technical side requires integration between Jenkins and Subversion. In this context, integration means that Jenkins could get any information attributes from Subversion and post common actions (COMMIT, MERGE, LOG etc.). Firstly, all needed tools for configuration management process should be extracted from PSAM model and, secondly, full services between different couples of tools should be created. For example, it could be services "Jenkins <-> JIRA", "Jenkins <->Subversion", "Jenkins<->Git" etc. If such services are created, the implementation of particular actions from the PSAM model could be easier because integration between different tools already exists and ready functions could be used. For example, one of the functions in service "Jenkins <->Subversion" could be "SVN_MERGE" that allows merging two different trees of a source code together. This function could be used in any project, where continuous integration server is Jenkins, and the Subversion system controls changes in a source code. Only values of function parameters should be different.

B. Lesson 2

Fails in experiments for project "3", provided in Table I, show strong dependencies between branching strategies and general version control. Implementation of common operations for version control, such as "commit", "merge" should be independent of general branching strategy. Environment model has to provide information on the kind of a source code branching strategy that should be used for a particular project. New model-driven approach should be extended by a new kind of models – source code branching model. This model should be dependent on the Environment Model.

C. Lesson 3

Environment model should be extended by options to model detailed infrastructure of particular environments. From the perspective of a customer, it could be better to obtain from the Environment Model such information as servers, firewalls, needed resources (RAM, storage etc.).

D. Lesson 4

Software configuration management plan should be generated from models of a new model-driven approach. In current implementation of a new approach, the generation of configuration management plan is manual and connection between document and models is not clear. The approach should be implemented by an ability to generate a configuration management plan automatically.

E. Lesson 5

Development or refactoring of any solution should be controlled by a version control system. It means that general framework for configuration management should also be managed. All scripts, functions, tool installation guides should be under a version control and all changes should be managed, for example, by bug tracking systems. All solutions should be parameterized and independent of other solutions. No any hardcodes should be in scripts.

VII. CONCLUSION AND FURTHER WORK

The paper provides a new model-based approach to the implementation of software configuration management. Unlike other model-based solutions to configuration management, a current approach does not require using any special technologies or tools but allows improving the existing solutions. New approach is also related to all sub-tasks of configuration management, not only to some particular tasks. The article also provides criteria to evaluate gains of a new approach. Based on the developed criteria, experiments have been made in five different projects. As a result, some practical lessons have been learned and described. These lessons could improve not only a model-driven approach provided in this paper, but also could improve some other solutions for configuration management. New approach is described without technical details of implementation because the number of pages in this paper is limited. Positive aspect of this fact is that abstraction could generate other ideas how to implement models of this approach.

One of the important directions of further work is related to the improvement of current approach according to the lessons learned. Two new models should be implemented and tested: "Tool Service Model" and "Source Code Branching Model". The framework for a configuration management plan should also be developed to generate it automatically from models.

References

[1] R. Aiello, *Configuration Management Best Practices: Practical Methods that Work in the Real World*, 1st ed. Addison-Wesley, 2010, p. 272.

[2] S.P. Berczuk and B. Appleton, *Software Configuration Management Patterns: Effective TeamWork, Practical Integration*, 1st ed. Addison-Wesley, 2003, p. 218.

[3] R. Calhau, R. Falbo, "A Configuration Management Task Ontology for Semantic Integration," *Proceedings of the 27th Annual ACM Symposium on Applied Computing* pp. 348–353 ACM New York, NY, USA, 2012. http://dx.doi.org/10.1145/2245276.2245344

[4] H. Giese, A. Seibel, T. Vogel, "A Model-Driven Configuration Management System for Advanced IT Service Management, " Available at: http://www.hpi.unipotsdam.de/giese/gforge/publications/pdf/GSV-MRT09_paper_7.pdf, 2009.

[5] O. Nikiforova, N. Pavlova, K. Gusarovs, O. Gorbiks, J. Vorotilovs, A. Zaharovs, D. Umanovskis, J. Sejans, "Development of the Tool for Transformation of The Two-Hemisphere Model to The UML Class Diagram: Tehnical Solutions and Lessons Learned," *Proceedings of the 5-th International Scientific Conference „Applied Information and Communication Tehnologies"*, 2012, Jelgava, Latvia, pp. 11–19.

[6] J. Osis, E. Asnina, *Model-Driven Domain Analysis and Software Development: Architectures and Functions*. IGI Global, Hershey – New York, 2011, p. 514. http://dx.doi.org/10.4018/978-1-61692-874-2

[7] W. Pindhofer, "Model Driven Configuration Management. Master work of Wien University," Wien, 2009.

[8] Y. Udovichenko, Upravlenije projektami ili kessonnaja boleznj projektov. (russian) [Online]. Available: http://experience.openquality.ru/software-configuration-management/, 2011.

[9] S. Dart, "The Past, Present, and Future of Configuration Management," CMU/SEI-92-TR-8, pp. 1–25., 1992.

[10] CMCrossroads, "How Configuration Management Is Changing: An Interview with Joe Townsend," April 8, 2014. [Online]. Available: http://www.cmcrossroads.com/interview/how-configuration-management-changing-interview-joe-townsend?page=0%2C0. [Accessed 02 September 2014].

[11] R. Yongchang, "Fuzzy Decision Analysis of the Software Configuration Management Tools Selection," in *ISISE '10 Proc. of the 2010 3rd Int. Symp. on Information Science and Engineering*, France, 19–23 June, 2010. ACM. pp. 295–297. http://dx.doi.org/10.1109/ISISE.2010.112

[12] J. de Almeida Monte-Mor, "GALO: A Semantic Method for Software Configuration Management," in *11th Int. Conf. on Information Technology: New Generations (ITNG)*, 2014. Las Vegas, USA, 7–9 April, 2014. ITNG: IOT360, pp. 33–39. http://dx.doi.org/10.1109/ITNG.2014.66

[13] Z. Toth, "Using Version Control History to Follow the Changes of Source Code Elements," in *17th European Conf. on Software Maintenance and Reengineering (CSMR), 2013*. Italy, March 5–8, 2013. IEEE Digital Library. pp. 319–322. http://dx.doi.org/10.1109/CSMR.2013.40

[14] J. Estublier, "Software configuration management: a roadmap," in *ICSE '00 Conf. on The Future of Software Engineering.* New York, USA, June 4–11, 2000. ACM. 279–289. http://dx.doi.org/10.1145/336512.336576

[15] Li. Ruan, "A new configuration management model for software based on distributed components and layered architecture," in *Proc. of the 4th Int. Conf. on Parallel and Distributed Computing, Applications and Technologies*, 2003. China, Aug. 27–29, 2003. IEEE Digital Library: IEEE. pp. 665–669, 2003. http://dx.doi.org/10.1109/PDCAT.2003.1236387

[16] M. Mingzhi, "A New Component-Based Configuration Management 3C Model and its Realization," in *Int. Symp. on Information Science and Engineering, 2008.* China, Dec. 20–22, 2008. IEEE Digital Library: IEEE. pp. 258–262. http://dx.doi.org/10.1109/ISISE.2008.244

Arturs Bartusevics currently is a Doctoral Student at Riga Technical University, the Faculty of Computer Science and Information Technology, the Institute of Applied Computer Systems. He obtained BSc (2008) and MSc (2011) degrees in Computer Science and Information Technology, respectively, from Riga Technical University. His research areas are software configuration management, release building and management process and its optimization. He works at Ltd. Tieto Latvia as a Software Configuration Manager.

E-mail: arturik16@inbox.lv

Leonids Novickis is a Head of the Division of Applied Systems Software. He obtained Dr. sc. ing. degree in 1980 and Dr. habil. sc. ing. degree in 1990 from the Latvian Academy of Sciences. He is the author of 180 publications. Since 1994, he has been regularly involved in different EU-funded projects: AMCAI (INCO COPERNICUS, 1995-1997) – WP leader; DAMAC-HP (INCO2, 1998-2000), BALTPORTS-IT (FP5, 2001-2003), eLOGMAR-M (FP6, 2004-2006) – scientific coordinator; IST4Balt (FP6, 2004-2007), UNITE (FP6, 2006-2008) and BONITA (INTERREG, 2008-2012) – RTU coordinator; LOGIS, LOGIS-Mobile and SocSimNet (Leonardo da Vinci) – partner. He was an independent expert of IST and Research for SMEs in FP6 and FP7. He is a corresponding member of the Latvian Academy of Sciences and an elected expert of the Latvian Council of Science. His research fields include Web-based applied software system development, business process modeling, e-learning and e-logistics.

E-mail: leonids.novickis@rtu.lv, lnovickis@gmail.com

Stefan Leye. He obtained a Diploma in Mechanical Engineering (2011). Primary field of study – integrated product development. He currently works at the Fraunhofer Institute for Factory Operation and Automation IFF, Magdeburg – Germany, as a Project Manager in the business unit Virtual Interactive Training. Main field of research: Virtual Reality and Digital Engineering solutions for product development and training.

Address: Sandtorstraße 22, 39106 Magdeburg, Germany;
Phone: +49 391 4090 114, fax: +49 391 4090 115;
E-mail: stefan.leye@iff.fraunhofer.de