doi: 10.1515/itms-2014-0004
Information Technology and Management Science
2014 / 17

Towards the Model-driven Software Configuration Management Process

Arturs Bartusevics [1], Leonids Novickis [2], [1, 2] *Riga Technical University*

Abstract **– Software configuration management is one of the most important disciplines within the software development project, which helps control the software evolution process and allows including into the end product only tested and validated changes. To achieve this, software management completes certain tasks. Concrete tools are used for technical implementation of tasks, such as version control systems, servers of continuous integration, compilers, etc. There are situations when some tools or scripts are implemented to support small particular parts of complex software configuration management process without a general picture of total process. Maintenance of such solutions is complex and more expensive than it could be. The study describes a model-driven configuration management approach, which foresees the development of an abstract environment model for the configuration management process that later is transformed to lower abstraction level models and tools are indicated to support the technical process. This approach allows implementing all tools and solutions according to the planned configuration management process and developing reusable solutions.**

Keywords **– Configuration management model, model-driven approach, software configuration management.**

I. Introduction

Software Configuration Management is a discipline that controls the evolution of software development process. To achieve this goal, software configuration management deals with the tasks such as configuration item identification, version control, configuration status accounting and configuration audit [1], [3], [4]. Software configuration management is described in ISO quality standards, where it is noted that the results of the process are needed for other sub-processes in software development [3]. General recommendations for configuration management process described in the quality standards are taken as the basis for many information technology companies worldwide. As it is noted in the studies [1] and [4], there are mainly two ways to achieve the standards. The first one is focused on the development of methodology and describes particular activities to be taken to implement configuration management. The second one is focused on the development of tools that most commonly deal with only one configuration management task.

One of the main problems of software configuration management implementation is the fact that implementing strategy of initial phase of software development process is not well thought out. However, a well-thought-out and realized configuration management strategy is the first condition which suggests that configuration management process has been implemented [1], [3], [4]. Due to the fact that configuration management implementation strategy is not well thought out and there are no specific activities, they start solving real tasks in the project. For example, there is a source code and it is submitted to version control, but there are no baseline strategies for development and maintenance. It is also not thought out how to organize parallel development and maintenance of parallel branches, but in case of project development such a need may arise. This is a situation that develops solutions to specific situations that arise in practice, but the solution does not necessarily correspond to the process of a general concept. Such solutions are difficult to maintain because they are eventually complemented with multiple corrections and adjustments, often resulting in dependency on another task solutions. It is the so-called "master factor" situation, when only one or a few people know how the particular software configuration management solution works, and it takes them a lot of time to find solution maintenance [1].

The second problem is related to the reuse of solutions for configuration management. If configuration management strategy has not been developed and thought out, every project of information technology company needs to develop configuration management solutions from scratch, because previous project solutions were purposed for dealing with too specific tasks and it is difficult to find solutions to concrete tasks there. Sometimes it is easier to implement a solution from scratch than to choose the existing one as basis.

The problems mentioned above additionally stimulate creating many tools that are focused on specific configuration management solution and tool manufacturers promise to solve all potential problems with a particular tool. However, both configuration management experts [1], [4] and quality standards [3] emphasize the need to choose a configuration management plan, which defines specific tasks, and only then to choose the tools that implement these tasks.

Configuration management experts [1], [4], propose at the initial stage to create a model of configuration management process, then to determine activities which are necessary for process implementation, and only then to choose the appropriate tools. Thus, the article first analyzes the existing solutions based on models and their disadvantages. Then it is offered a new model-driven configuration management solution that suggests specific activities for configuration management process plan and its implementation. The final part of the article defines further research directions as well as expected benefits of the designed solution.

II. Related Research

Initially, the model-oriented solutions appeared in the studies, where only one of the main configuration management tasks was solved, for example, configuration item identification [10], [2], [13], [6].

In article [10], the authors propose making developed product modeling. At first, it is emphasized that by defining potential candidates of configuration items, not only a source code should be taken into account, but also the documentation that describes the product from different points of view. Modeling approach is based on a gradual system of tree-like structure development.

While researching version of controlled solutions, one may encounter model creation [2], [5]. Study [2] describes an algorithm, which consists of meta-model controlled items. The meta-model describes the syntax of the model, which could be exposed to version control. The algorithm is intended for version control systems that control models, not a source code, which is specific to projects that use a model-driven development approach. Thanks to meta-model, it is possible not only to manage the change of models, but also to make the merge of different model versions and to see merge conflicts, like it can be seen in source code file change combination. A similar solution is offered in the following source [5]. There is mentioned the fact that the version control and parallel development may not be limited by source code file control. The projects, which use a model-driven approach, need to control the model versions, while the vast majority of existing version control systems fails to maintain versions of the models. Therefore, the authors of study [5] propose an approach that requires the development of an abstract model for version control system.

Model-driven approach is used for building and installation processes in the framework of configuration management [13], [6]. The following article [13] is not about specific tools, but about an abstract model, which describes the building and installation process. The solution offers a model that describes the activities necessary to implement in the building and installation process. However, the following article [6] emphasizes the importance of a correct choice of configuration items for qualitative product building that should be included in the building process. Article [6] proposes a model-based methodology, which provides a set of configuration items to be included in the building process. Methodology uses heuristics and analyzes all possible configuration item sets that could potentially be included in the building process.

There are model-based solutions that are focused not on one, but numerous configuration management tasks [12], [7], [8]. In order to have a better idea about solution maturity and disadvantages in the context of a model-driven approach, general principles of model-driven approaches [11], [9] used in software development have been studied. Model-driven solutions should have the following features:

- The existence of a meta-model. The solution must have the source from where one can create a configuration management process model.
- Different levels of abstraction models. Model-driven solution should have an abstract model that describes a configuration management process, independently from computing and tools that will be used in the process of implementation. However, there should be other types of models, where you can see signs of computing, platform specificity and tool role in the process.
- Relationship between different types of models and transformation rules, which allows the model with one abstraction level to transform into the model of another abstraction level. Transformation rules should be formalized in order to be subjected to computer processing.
- Tool support. In order not to leave the solution at a theoretical level, we need tools that are able to create models from the existing meta-models and to make necessary transformation.
- Relationship with a problematic environment. This relationship is needed for a viable solution. Otherwise, the created and implemented model is unable to respond to changes in the problematic environment and it will be impossible to quickly determine weaknesses in the model. Over time, this can lead to specific tool configuration or code corrections, implementation process and the process model will lose its relevance.

Within the following solution [12], the authors have developed configuration management and model-driven development unification concept, meta-model, which allows creating an abstract model of product configuration, Eclipse Modeling Framework tools, which allow obtaining a specific configuration model from an abstract model platform, as well as instructions for expanding the created methodology and tools. Although the solution corresponds to the main principles of a model-driven concept, it is focused on projects where development occurs based on model-driven approaches. Meta-model is only a configuration item for task identification.

Study [8] presents a methodology, which considers a configuration management process as a whole process. Configuration management principles for solution creation are taken from ITIL standard (Information Technology Infrastructure Library) and later abstract models are created, from which one can create an abstract configuration management process, and later this model can be transformed into a platform interdependent model. The approach uses the main principles of model-driven development. The meta-model based models offer the necessary abstraction that improves the process of configuration management, transparency and allows the user, if necessary, to implement a model of a particular technology, for example, to make model transformation. A system prototype is created that implements model-driven configuration management. The author of article

[8] provides extensive research further, in order to arrive at a particular solution, which is also recognized in the conclusions. The solution is focused only on one technology (JAVA). Thus, it is difficult to assess the possibility of practical application, and an opportunity to realize the relationship with the problem is also not analyzed.

Solution [7] offers an approach to various configuration management tools for mutual integration. In order to maintain a complete configuration management process, a number of tools are required: version control system, problem management system, building servers, continuous integration servers and many other tools. In practice, all the above-mentioned tools work separately from each other. In order to facilitate a configuration management process, an approach has been offered to integrate all these tools together. However, in order to integrate various configuration management tools, it is necessary to define a general concept of each integrated tool [7]. The article offers configuration management task ontology. This ontology is used as a configuration management model that shows how various configuration management tools are integrated. Ontology is mainly based on the version control, which is one of the key concepts of configuration management. Ontology provides information about configuration management of sub-reciprocal links, which are expressed in concepts, links, tasks, agents, output and input data sets [7]. There is no guidance how ontology can be used in specific project configuration management. In terms of relationship between problems, it is not clear how to make changes in ontology, which ontology editors it is recommended to use and how to determine the moment when these changes have to be made.

Solution [14] intends to apply the fuzzy logic theory to create a multi-criteria decision-making system. Decision-making main objective is to determine the optimal set of configuration items that will be subjected to a configuration management process.

III. General Approach to Model-driven Software Configuration Management

This article proposes a new model-driven configuration management implementation approach, which unlike other model-driven solutions is focused on a gradual process rather than solving a specific task. Another difference is that the solution does not impose any specific technology or tool, but suggests using the existing solutions by increasing the reuse.

During the development of configuration management model-driven approaches, the main ideas of MDA [9], [11] were taken into account. A solution should contain abstraction models with different levels and clearly defined rules how to move from one abstraction level model to another.

The highest level of abstraction MDA is CIM (computation independent model). While researching configuration management definitions, it was searched for something, which could build a configuration management model with the highest level of abstraction.

Summing up definitions [1], [3], [4], the authors managed to come to a conclusion that configuration management in its highest level of abstraction answers the following question: “How to transfer software performed changes from one instance to another at the right moment?” A key highlight of this subject is placed on the changes that were made to the developed product and its ability to transfer changes from one instance to another, because usually several instances are used in a software development project [1].

Thus, creating a configuration management model with the highest level of abstraction, the following aspects should be taken into account: the existing instances, and the way, software development changes will go starting from development and finishing with exploitation, as well as which versions are formed for the developed product. This type of model, developed in the framework of the solution, is called an environment model and will be hereinafter denoted as an EM.

Each model must have the source or meta-model; therefore, in the developed concept the element “Meta-model of EM” has been introduced. At the initial stage, a configuration manager, who is planning the process, is proposed to make an EM using elements from a meta-model.

Expert system knowledge base represents rules, which are able to determine general configuration management risks depending on the state of EM. Such rules hereinafter will be marked as “Rules of Risks, Compilation”. The expert system within the developed concept is indicated as “Expert System”. If a configuration manager creates an environment model that does not meet basic principles of configuration management [1], [4], an expert system then reports about potential risks of the model. As soon as a configuration management process EM has been obtained and an expert system has identified risks, it is necessary to form a model with a lower level of abstraction. The essence of a model has been defined in the following article [7], which says that regardless of the planned configuration management process, in the end it supports a variety of tools which carry out certain activities to implement source code management, compilation, building activity etc., performs the basic tasks of configuration management. Environment Model shows the further way of software changes. Taking into account the following opinion [7], it would be logical at this stage to understand what kind of action should be taken to implement an EM described way in accordance with principles of configuration management [1], [4]. The following information [11] should also be taken into account: a CIM model is followed by a platform-independent model (PIM), and actions at this stage cannot be attracted to specific technologies and platforms. Thus, the concept has introduced the element “Platform Independent Action Model” that is marked as PIAM. This model has also a specific source or meta-model “Meta-model of PAM”.

A new element “E→P” appears in an expert system or the transformation rules from an Environment Model to a Platform Independent Action Model. Thus, obtaining an

Environment Model, a configuration manager uses "E→P" transformation rules to get a PIAM model. PIAM meta-model defines global activities, which deal with configuration management tasks.

Once a configuration management PIAM model has been created, a configuration manager chooses a solution from a "Solutions database" to each action of PIAM. Thus, it becomes known how PIAM model specific items will be implemented in each task. This model is called "Platform Specific Action Model", and is marked as a PSAM. Fig. 1 shows a general scheme of model-driven configuration management. Arrows with numbers represent steps of approach.

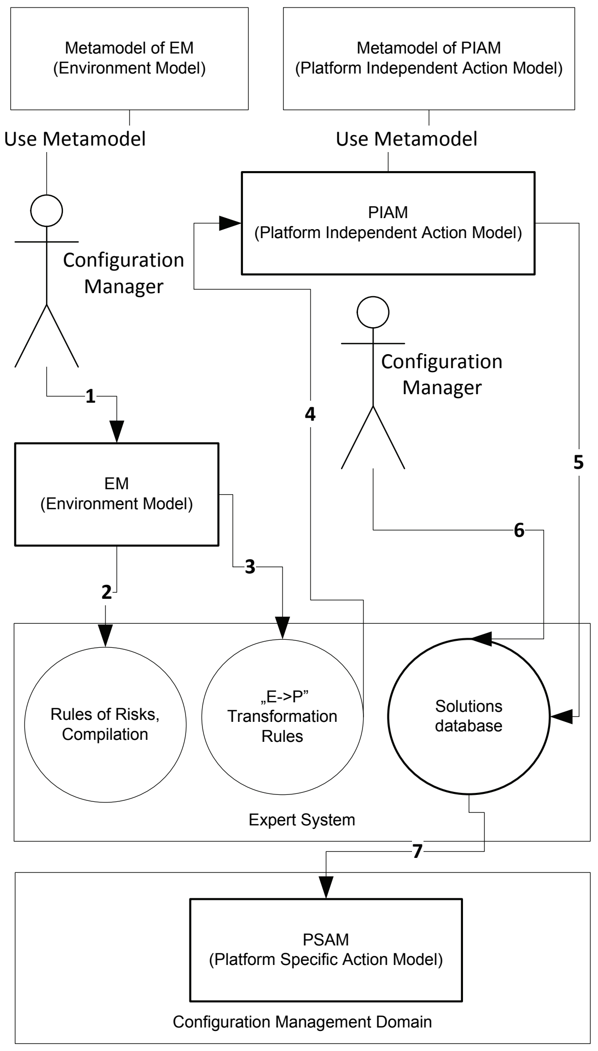
Metamodel of EM
(Environment Model)
Metamodel of PIAM
(Platform Independent Action Model)
Use Metamodel
Use Metamodel
Configuration Manager
PIAM
(Platform Independent Action Model)
Configuration Manager
1
4
5
EM
(Environment Model)
6
2
3
Rules of Risks, Compilation
„E->P" Transformation Rules
Solutions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

Fig. 1. Conception of model-based configuration management.

A detailed description of elements of the model-driven approach for the implementation of configuration management process is provided below:

EM (Environment Model) – Displays the flow of configuration items in the context of different instances. For example, the development and maintenance of development project displays how changes come from the development environment to different test environments, and finally, to the operation or production environment. It also shows how new versions of configuration items are created and in what order these versions are transferred from one instance (environment) to another.

Metamodel of EM – Source for an Environment Model. It contains all the components of which EM may consist. During the EM modeling course, the user can take components only from a meta-model.

PIAM – shows which actions are needed to ensure the implementation of the flow of EM model configuration elements through the environments. This model has only abstract actions. For example, there is an element "PREPARE_BASELINE", but there is no information about platform, technology, manufacturers etc.

Metamodel of PIAM – Source of PIAM model. It contains all the components of which PIAM may consist.

PSAM – PIAM extension model, which eliminates abstraction from technologies, platforms, manufacturers, and etc. All actions have specifically selected solutions from a solution database, so that the model contains certain information necessary for activity implementation.

Configuration Management Domain – an environment, where a configuration management system operates: tools, interaction, project factors that affect the configuration management process etc.

Rules of Risks, Compilation – Rules, which determine problem areas in an EM, depending on the problem characteristics and model component condition. These rules also contain different types of recommendations to prevent one problem or another, as well as how to react to changes in a problematic environment (changing one or more parameters).

E→P – rules that determine how to get a PIAM from an EM.

Solutions database – a database of solutions of actions from a PIAM.

Expert System – an expert system that combines configuration management solution database and transformation rules from an EM to a PIAM, as well as "Rules of Risks, Compilation".

IV. Practical Implemetation of a Model-driven Approach of Configuration Management

EM consists of the following elements:

a) **Environment** – instance that contains configuration elements that make up the software product,

b) **ConfigurationItemFlow** – a flow, through which configuration items are transferred from one environment to another,

c) **Event** – consists of one or more configuration flows. Event may contain several flows, because sometimes, before transferring configuration to some environment, at first it is transferred to the appropriate environment copy to make sure that transferring will occur correctly. For example, in order not to jeopardize the test environment, at first configuration may be transferred to the test environment copy and then to the real test environment. In this case, one event will be responsible for transferring changes from the development environment to the test environment, and this event includes two configuration flows: one flow transfers the changes to the copy of the test environment, and in case of success, the second flow is triggered, which transfers the same configuration to the real test environment.

Once an EM has been created and all the environments have been identified, all the events and flows can create a PIAM, which at the highest level includes the element "Continuous Integration Server" – a centralized location, from where all the events in configuration transfer between environments will be managed. Continuous Integration Server will contain all the events of EM with all relevant configuration flows. Then "E→P" transformation rules will be used, which are based on configuration management practices, standards or frameworks [1], [3], [4], and for each configuration flow activities from the following action set will be determined:

DEVELOPMENT – a process of development is included in a PIAM model, as configuration management controls what kind of development regulations should exist that control principles of configuration item development and changes [1], [3]. By simulating these activities, it is possible to determine, for example, certain technology and principles of configuration item development of a certain project, procedure that requires changes will be agreed, tools for performing change management, regulation documents, instructions, etc. It should always be kept in mind that if the development process is not controlled and managed, further steps in the configuration management process will be difficult to predict [1].

COMMIT_CHANGES – an activity that determines preservation of change development in a version control system. Activity is controlled by a version control task. All developed product configuration items should be subjected to version control, in order to be able to always see when, who and why made changes [1], [3], [4]. This activity is modeling a version control of a certain project and specific rules to be followed by developers, saving changes in a version control system. Moreover, tools that automatically monitor compliance with required rules and principles can be recorded into activity attribute.

PREPARE_BASELINE – this activity, depending on a version control system, controls the procedure and technical support of environmental baseline preparation. Configuration management controls how to define a baseline to any changes, against which the changes will be made [1]. Environment model assumes that each environment has its own baseline – an approved condition of product source code that corresponds to environment configuration. Thus, before any configuration transmission from one environment to another, it is necessary at first to rebuild a source code of the relevant environment. In practice, this activity provides the transfer of changes between two branches of version control system, which is also called the merge. This activity describes the process of configuration transfer between two repository branches and provides a description of tools that support it technically.

COMPILE_BUILD – an activity that builds a product from a corresponding source code. Compilation and the build should result in the creation of an executable item.

INSTALL_BUILD – an activity that installs the built product into a certain environment (instance).

PRODUCT_DELIVERY – an activity that prepares and ships the built product to a customer. This activity is necessary in order to be able to install the product into the environment, which is supported by a customer, and a developer team does not have access, for example, to a production environment. Activity is modeling a procedure, which is necessary to follow while preparing the product for shipment. For example, together with the built product it is necessary to send the description of the version and instructions how to install the product into instance; these requirements control configuration management and quality standards or frameworks [1], [3], [4].

ENV_UPDATE_NOTIFICATION – an activity that sends a signal to the development team that environment is updated by new version of product. At this moment, the development team should record the fact, and accordingly supplement configuration item information in a version control system, as well as install the same version of the product into all environments, which is a test copy of the customer's environment. This process needs an obligatory renewal of the appropriate environmental baseline. This action controls the procedure that is mandatory to comply by receiving information from a customer about any environment restoring.

Each PIAM configuration flow for each activity contains the following attributes, which will not be completed, because at this stage a specific solution to any activity has not been chosen yet:

Platform – a platform specific name, which is necessary for activity implementation. This attribute is necessary because one activity may have multiple solutions across multiple platforms, for example, a script which transmits changes between two repository branches can be implemented in both Linux and Windows platforms.

SolutionName – a unique solution name to differentiate and manage it.

NeededTools – tools needed for technical implementation of the solution.

LocationsOfSolutions – this attribute is intended to store any information about the existing solutions for a certain action. There, for example, the script names, location of special programs, instruction etc. can be stored.

Description – additional description for a particular action in the context of the project or solution that could provide additional information to a configuration manager.

Once a PIAM is ready, a configuration manager chooses a solution to each configuration management activity from "Solution Database". If an appropriate solution is missing, then it should be developed by putting information into a solution database and only then the solution should be functioning. As soon as all the activity attributes are filled, thanks to a solution database, a PIAM becomes a PSAM, which is an extended version of platform-independent action model. In order to begin process model functioning in the problem environment, it remains to realize the selected solutions for each activity. Fig. 2 shows a visual example of EM, PIAM and PSAM.

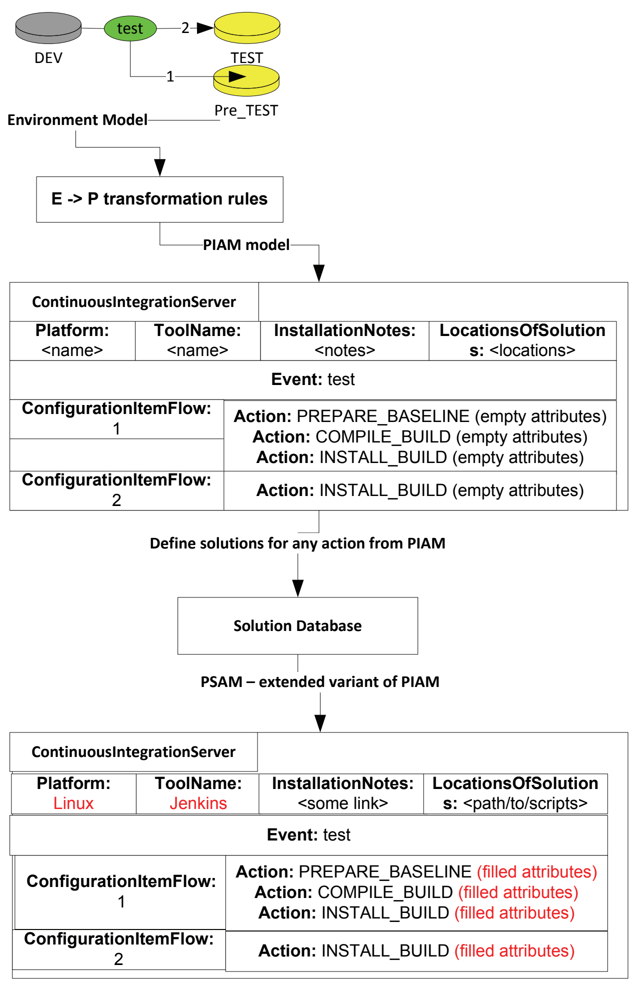
DEV
test
2
TEST
1
Pre_TEST
Environment Model
E -> P transformation rules
PIAM model
ContinuousIntegrationServer
Platform: <name>
ToolName: <name>
InstallationNotes: <notes>
LocationsOfSolutions: <locations>
Event: test
ConfigurationItemFlow: 1
Action: PREPARE_BASELINE (empty attributes)
Action: COMPILE_BUILD (empty attributes)
Action: INSTALL_BUILD (empty attributes)
ConfigurationItemFlow: 2
Action: INSTALL_BUILD (empty attributes)
Define solutions for any action from PIAM
Solution Database
PSAM – extended variant of PIAM
ContinuousIntegrationServer
Platform: Linux
ToolName: Jenkins
InstallationNotes: <some link>
LocationsOfSolutions: <path/to/scripts>
Event: test
ConfigurationItemFlow: 1
Action: PREPARE_BASELINE (filled attributes)
Action: COMPILE_BUILD (filled attributes)
Action: INSTALL_BUILD (filled attributes)
ConfigurationItemFlow: 2
Action: INSTALL_BUILD (filled attributes)

Fig. 2. Example of configuration management models.

In the example, shown in Fig. 2, an environment model is implemented with three environments: DEV, TEST and Pre_TEST. The task is to transfer configuration from a DEV to TEST environment, and it is the responsibility of the Event “test”. ConfigurationItemFlow “1” restores a source code in a version control system that meets a TEST environment, compiles the restored source code, and installs it into a Pre_TEST environment. If the process has been successful, the ConfigurationItemFlow “2” is implemented, which installs into a TEST environment the same installation package, which comes from the first flow COMPILE_BUILD action. Thus, the Event “test” has met the main goal and has transferred configuration from a DEV to TEST instance. To start PSAM functioning in a problem environment, it remains to implement the selected solution activities PREPARE_BASELINE, COMPILE_BUILD and INSTALL_BUILD.

V. Results, Further Studies and Expected Benefits

Based on studies of configuration management solutions and solution improvement tendency, a new model-driven approach has been created for implementation of configuration management. Within the framework of the research, meta-models for EM and PIAM models for a new approach have been developed.

Further studies should develop the prototype of proposed approaches to automate the modeling and transformation process. The results of experiments should determine the weaknesses of the current solutions, as well as develop criteria, which will help to determine whether the solution meets quality standards.

The main expected benefit of the proposed solution is the opportunity to see common configuration management processes and to track compliance with the technical implementation. Thanks to a PSAM that provides information on what tools are needed and how they exchange information, there is an opportunity to automate the process. Thanks to the model, there is an opportunity to configure tools in compliance with specific process goals, rather than to correct certain errors or solve only one of the tasks. This allows spending resources for tool configuration more efficiently, preventing not a specific error, but its cause, as well as it minimizes the risk of the error occurrence in the future. Another benefit could be the process transparency. If a new person joins the project, he has a chance to get acquainted with a configuration management process in a whole and understand how and for what purposes the tools operate. Process transparency will allow if necessary making changes in tool configuration more effectively. The main goal of the solution is to improve the quality of configuration management process, its transparency, as well as increase the efficiency of tool configuration and reduce the maintenance costs, so that there will be more opportunities to eliminate the causes of error, but not the error itself.

References

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

[2] K. Altmanninger, “Models in conflict – towards a semantically enhanced version control system for models,” Lecture Notes in Computer Science, LNCS, 2008; vol. 5002. pp. 293–304. http://dx.doi.org/10.1007/978-3-540-69073-3_31

[3] R. Bamford, J. Deibler, *Configuration Management and ISO 9001:* SSQC, 1999.

[4] A. Berczuk, *Software Configuration Management Patterns: Effective TeamWork, Practical Integration,* 1st ed., Addison-Wesley, 2003.

[5] T. Buchmann, A. Dotor, B. Westfechtel, “Model-Driven Development of Software Configuration Management Systems,” *4th International Conference on Software and Data Technologies*, ICSOFT, 2009, pp. 309–316.

[6] O. Bushehrian, “Automatic object deployment for software performance enhancement.” *The Institution of Engineering and Technology*, vol. 5, Iss. 4, pp. 375–384, 2011.

[7] R. Calhau, R. Falbo, “A Configuration Management Task Ontology for Semantic Integration,” *Proceedings of the 27th Annual ACM*

Symposium on Applied Computing, ACM, New York, NY, USA, 2012, pp. 348–353. http://dx.doi.org/10.1145/2245276.2245344

[8] H. Giese, A. Seibel, T. Vogel, A Model-Driven Configuration Management System for Advanced IT Service Management, 2009. [Online]. Available: http://www.hpi.unipotsdam.de/giese/gforge/publications/pdf/GSV-MRT09_paper_7.pdf [Accessed: Sept. 15, 2014].

[9] 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 5th International Scientific Conference „Applied Information and Communication Tehnologies”,* 2012, Jelgava, Latvia, pp. 11–19.

[10] Oject-Oriented Software Engineering Using UML, Patters and JAVA “Software Configuration Management”, 2002. [Online]. Available: http://www.bilkent.edu.tr/~bakporay/cs_413/Bruegge_L28_ConfigurationManagement_ch12lect1.ppt [Accessed: Sept. 28, 2014].

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

[12] W. Pindhofer, “Model Driven Configuration Management,” Master thesis, Wien University, Wien, 2009.

[13] P. N. Sindhuja, G. Surajit, “Software Deployment: Concepts and Technologies,” *ICFAI Journal of Systems Management*, 2008.

[14] J. Wanga, Lin Yung-I, “A fuzzy multicriteria group decision making approach to select configuration items for software development,” MathematicsWEB, *Fuzzy Sets and Systems*, 2002, pp. 343-363.

[15] Y. Udovichenko, Upravlenije projektami ili kessonnaja boleznj projektov, 2011. [Online]. Available: http://experience.openquality.ru/software-configuration-management [Accessed: Aug. 10, 2014].

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 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, 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: lnovickis@gmail.com.

Artūrs Bartusevičs, Leonīds Novickis. Modeļu vadāmais programmatūras konfigurācijas pārvaldības process
Programmatūras konfigurācijas pārvaldība ir viena no svarīgākajām programmatūras izstrādes disciplīnām, kas kontrolē produkta evolūciju un nodrošina to, lai produkta gala versijā būtu tikai kvalitatīvas un notestētas komponentes. Lai to nodrošinātu, konfigurācijas pārvaldība veic vairākus uzdevumus: konfigurācijas vienumu identifikāciju, versiju kontroli, būvējumu un instalāciju pārvaldību, konfigurācijas vienumu uzskaiti. Šo uzdevumu risināšanai tiek izmantoti konkrēti rīki, kas implementē uzdevumus tehniski. Bieži ir situācijas, kad konkrēts rīks ir implementēts, lai risinātu konkrētu mazu problēmu vai uzdevumu, taču pilnīga konfigurācijas pārvaldības procesa pārskata nav. Tādā veidā implementētu risinājumu ir ļoti grūti uzturēt, tas ātri apaug ar labojumiem un pielāgojumiem, un to vairs nav iespējams pielietot citā projektā bez papildu modifikācijām. Līdz ar to sanāk patērēt neracionāli daudz līdzekļu konfigurācijas pārvaldības procesu atbalstam. Raksts piedāvā jaunu modeļvadāmu pieeju konfigurācijas pārvaldības ieviešanai. Jaunās pieejas ietvaros tiek piedāvāts izveidot abstraktu vides modeli un vēlāk to transformēt cita veida modeļos, pakāpeniski ieviešot risinājumu tehniskās detaļas. Jaunā metodoloģija ļauj ieviest konfigurācijas pārvaldības risinājumus atbilstoši plānam, atkārtoti izmantojot jau esošos risinājumus. Atšķirībā no citām līdzīgām metodoloģijām, šī pieeja neuzspiež kādu noteiktu rīku. Tas ļauj reorganizēt esošos risinājumus, palielinot iespēju izmantot tos atkārtoti. Raksta sākumā ir sniegta literatūras analīze, kuras ietvaros esošos modeļvadāmos risinājumus parāda konfigurācijas pārvaldībā. Tiek analizēti esošo risinājumu trūkumi. Turpinājumā jaunā pieeja tiek aprakstīta kopumā, dotas modeļu definīcijas. Raksta nobeigumā tiek dots modeļu vizuālais piemērs, aprakstīti turpmākie pētījumi un jaunā risinājuma sagaidāmie ieguvumi.

Артур Бартусевич, Леонид Новицкий. Процесс управления конфигурациями программного обеспечения, основанный на моделях
Управление конфигурациями программных продуктов является одной из самых важных дисциплин, которая контролирует эволюцию программных продуктов и обеспечивает вхождение только качественных и протестированных составляющих в конечную версию программного продукта. Для того, чтобы обеспечить данное условие, необходимо выполнить ряд задач: идентификация конфигурационных элементов, контроль версиями, управление сборками и инсталяциями продукта, учёт конфигурационных элементов. Для решения подобных задач используется ряд программ и различных самодельных приложений. Возникают ситуации, когда технические решения покрывают лишь узкий круг задач и проблем, в то время как полная картина процесса управления неизвестна. Такие решения являются очень специфическими для конкретных проектов, решения очень тяжело поддерживать, так как они обрастают исправлениями и обновлениями. Подобные решения нельзя использовать повторно в других проектах. Это приводит к тому, что на управление конфигурациями тратится нерационально много средств. Статья предлагает новый подход для внедрения процесса управления конфигурациями, основанного на моделях. Сначала предлагается составить абстрактную модель среды, постепенно трансформируя её в другие модели, поэтапно уточняя технические детали реализации. Новый подход не требует внедрения конкретных программ или технологий и позволяет реорганизовать имеющиеся решения для увеличения их повторного использования. В начале статьи анализируются другие подходы, основанные на моделях, их недостатки. Позже описывается новый подход в целом, даются определения моделям. В конце работы приводится визуальный пример моделей, ожидаемые преимущества, а также делаются выводы.