DE GRUYTER OPEN

Applied Computer Systems

doi: 10.1515/acss-2015-0008

2015/17

Models and Methods of Software Configuration Management

Arturs Bartusevics [1], Leonids Novickis [2], Stefan Leye [3],
[1, 2] *Riga Technical University, Latvia,* [3] *Fraunhofer IFF Institute, Germany*

***Abstract* – The paper provides collection of experience of developing models and methods for implementation of software configuration management process. Nowadays, during the age of agility, startup of new software development projects is like an explosion. Sometimes, in a few days a customer is willing to get the first release, but formalized and reusable process for its creating is not ready yet. Software configuration management controls the evolution of software items and supports building and deployment process for product releases. Models and methods for implementation of software configuration management are presented in the paper. Unlike other approaches, the provided models and methods are related to decrease software configuration management implementation time in new projects. The paper provides a simplified use case to illustrate practical application of the mentioned methods and models.**

***Keywords* – Model-driven approach, software configuration management.**

I. Introduction

Software configuration management is a discipline that controls software evolution process [1], [2]. Sometime, based on the above-mentioned definition, enterprises perceive software configuration management only as version control and source code management. In real software development projects, the scope of software configuration management is extended by different tasks, such as building and deployment management, integration with bug tracking systems, release management, integration with unit testing, continuous integration etc. To apply the mentioned tasks of software configuration management, enterprises use different tools, for example, Subversion, Git, Mercurial for version control and source code management, Jira for bug tracking, Jenkins, Bamboo, Serena for continuous integration, building and deployment management [3], [4] etc. Additionally, enterprises apply scripts and custom tools to automate the mentioned tasks and to integrate together different tools [3]. In the context of this paper, ready software configuration management process is an executable source code that operates with different tools to apply and automate a set of software configuration management tasks, such as version control, source code management, building and deployment management, release management, integration with continuous integration, unit testing and many other tasks related to software configuration management.

Usually enterprises have a vision about a set of tasks related to software configuration management. Enterprises also have tools, scripts and integration solutions to apply the mentioned tasks of software configuration management. The main challenge is how to implement existing tools, scripts and solutions in a new software development project as soon as possible. Sometimes, startup of new projects is like an explosion. It means that in a few days customer is willing to get the first version of product; however, the formalized and reusable process is not ready. It causes a situation that we called a "master factor", when the first release could be prepared from a local computer by a particular developer using only his personal practical experience but not a formalized and reusable process. It could be a reason of unexpected errors in software builds and unstable releases in production environments. Why is so much time required for implementation of software configuration management process in new projects? It is due to the fact that scripts and tools existing in other projects of particular enterprises are very specific for mentioned projects and not reusable. There is a lack of formalized scientific approach on how to increase the reusability level of existing solutions related to software configuration management.

A. Problem Formulation

In the current paper the following problems are underlined:

- Lack of formalized and scientific approaches on how to increase the reusability level of existing solutions of software configuration management tasks;
- Lack of approaches defining a formalized way from requirements of software configuration management to implementation in the context of executable source code.

Paper [5] provides ideas that problems mentioned before could be solved with a model-driven approach, because this approach separates requirements from implementation for particular platforms and gives a formalized way from requirements to implementation and allows generating an executable source code automatically. These options allow reducing implementation time of software configuration management that is the main challenge underlined in the current paper. Tracy Ragan [5] in her paper states that during huge improvement of cloud computing technologies, static scripts cannot solve any tasks of software configuration management, because paths, IP addresses are not known absolutely. Paper [6] provides practical evidence of the mentioned ideas: many model-driven tools are available to automate tasks of software configuration management. Usually, these tools relate only to building and deployment management and do not support other software configuration management tasks. Other disadvantage is that tools require

using a set of specific tools that could be unavailable or unreliable for a particular enterprise.

B. Scientific Novelty of Paper

- Model-driven approach for implementation of software configuration management. Unlike other approaches, this one is intended to decrease implementation time using existing tools and scripts that are trusted by a particular enterprise, do not impose some specific tools related to all tasks of software configuration management, not only to building and deployment management.
- Three new models for representation of software configuration management in the context of a new approach.

C. Structure of the Paper

The current paper contains three main sections and conclusions. The second section provides a brief introduction of other studies related to implementation of software configuration management by model-driven approaches. The third section contains the description of the provided model-driven approach, but the next one provides a simplified example that illustrates practical application of the provided approach, models and methods. Finally, conclusions and directions for further researches are provided.

II. Related Research

The current paper is not the first attempt to provide software configuration management as a complex process with multiple tasks, not only version control or deployment management. Study [3] provides an approach for integration of different tools to solve tasks of software configuration management. Authors [3] provide the ontology for definition of concepts of each tool related to software configuration management. The main advantage is strong formalism of the provided approach. There is a lack of recommendations on ontology editors that could be used for practical application of this approach. In addition, there are no recommendations on how to apply existing tools and scripts in the new approach.

Paper [4] provides an abstract model for complex software configuration management tasks. It is a positive aspect that the provided model is based on most popular quality standards and frameworks such as CMMI and ITIL. Authors [4] say that their research is at an initial stage and additional models should be developed to reduce the level of abstraction. In general, this study has the same disadvantage as [3]: there is a lack of recommendations and methods how to apply existing scripts, tools and solutions with the provided approach.

During development of a new model-driven approach provided in the current paper, many ideas were taken from [3], [4], [5] and [6]. From papers [3] and [4], the idea about integration of different tools was taken. It was concluded that a novel model-driven approach should describe many different tasks of software configuration management, not only version control or deployment management. Additionally, the new approach should allow adding, modifying and deleting tasks of software configuration management, because enterprises have private vision about a set of tasks that should be implemented with a software configuration management process.

The first results of new approach provided in the current paper have been described in [10]. It has been really the first attempt to describe software configuration management by different models which are connected together by a strong defined approach. Later, new models have been developed and described in [9], [8] and [7]. New models have been tested by practical experiments and lessons learned from the mentioned experiments described in [11]. Based on comments of reviewers of papers [9], [8] and [7] and practical results provided in [11], a new version of model-driven approach for software configuration management has been designed. New approach has improved models for representing a software configuration management process. Unlike previous related studies [9], [8], [7], [11], the current version of approach has the following differences:

- Model-driven approach is oriented to the complex process of software configuration management. Tasks of the mentioned process can be added and delated.
- The main scope of approach is increasing the reusability level of existing tools, scripts and solutions that already exist in a particular enterprise.
- Approach is aimed at decreasing the implementation time of software configuration management in new projects.
- Approach provides a method of how to store existing scripts and solutions to make them reusable.

III. Approach, Models and Methods of Software Configuration Management

During improvement of work described in [9], [8], [7], [11], the new approach has been designed for implementation of software configuration management. Provided approach expects that ready software configuration management process is an executable source code that can apply different tasks of the mentioned process. The approach requires that the mentioned source code should be reusable and executed only from a centralized software configuration management server. Usually, enterprises use one of continuous integration servers as a general software configuration management server, for example, Jenkins, Bamboo, CruiseControl etc. New model-driven approach provides generation of source code for software configuration management using a set of models and methods. General approach is illustrated in Fig. 1.

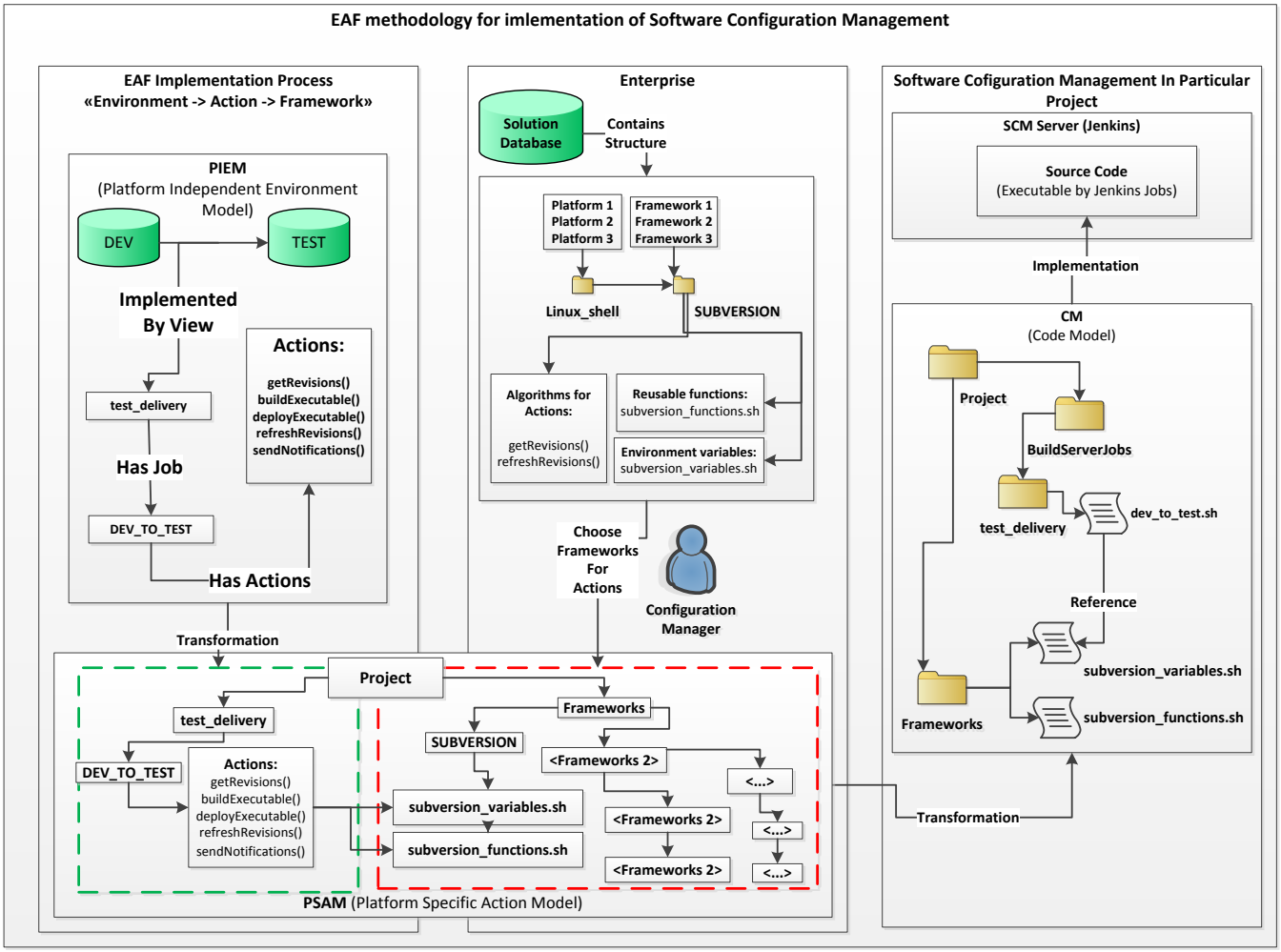

Fig. 1. Model-driven approach for implementation of software configuration management.

The provided approach illustrated in Fig. 1 contains three parts:

- *Software configuration management process in a particular project.* The process is a software configuration management server (SCM Server) and a source code (Source Code) for implementation of all software configuration management tasks in a particular project.
- *Solution Database.* Centralized database in a particular enterprise where all executable source code fragments for software configuration management tasks are stored. Database should be designed by a special method, provided by the current approach. At least one configuration manager should manage this database to add, modify or remove executable and reusable source code units for particular tasks of software configuration management.
- *Implementation process.* The main scope of this process is generation of source code for software configuration management using special models and Solution Database. Implementation process contains three models:
  - *PIEM (Platform Independent Environment Model).* This model represents all environments and actions needed to move software changes from one environment to the other. Environment in the context of this approach is a set of infrastructure, for example, database servers, application servers, firewalls, other dependent tools, in other word, all that needs to run software. Usually, software development projects have many environments with different scope, for example, DEV for development, TEST for testing and PROD for exploitation or production.
  - *PSAM (Platform Specific Action Model).* This is extended variant of PIEM model provided implementation details about actions described in a platform independent environment model. The model has two parts. The first part (marked with a green box in Fig. 1) represents the structure of actions needed to move software changes between environments. This structure should be copied from the PIEM model without any modifications. The second part (marked with a red box in Fig. 1) represents the frameworks needed for implementation of the mentioned actions. Frameworks should be chosen from the Solution Database by a configuration manager. Each job with actions has references for reusable functions of

particular frameworks and environment variables needed to implement framework in a software configuration management server.

- ○*CM (Code Model).* This model represents a structure of source code files and folders that implement a particular platform specific action model according to rules of platform and special programming language (in Fig. 1, an example with Linux Shell scripts is given). Code model should be generated automatically from a particular PSAM model. Then it could be implemented as a source code in a particular software configuration management server.

Provided approach (Fig. 1) is called EAF (Environment -> Actions -> Framework). This name illustrates the process of generation of a source code. Firstly, Environments should be defined. Secondly, Actions should be specified to move software changes between defined environments. Finally, Frameworks should be selected from the Solution Database to implement the mentioned actions in a particular platform.

IV. Solution Database

Solution Database is a warehouse where all executable source code units for tasks of software configuration management are stored. Solution Database should be designed by the method provided in the EAF approach. Fig. 2 provides the structure of Solution Database with an example for Linux platform and SUBVERSION framework, which works with the Subversion version control system.

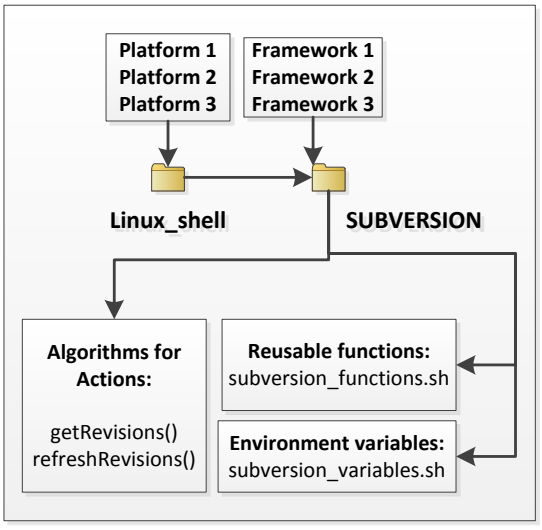

Fig. 2. Solution Database.

The structure of Solution Database provided in Fig. 2 contains all executable source code units for actions of software configuration management. The method of grouping mentioned solutions contains the following steps:

- Detect platform where a code unit should be executed. Fig. 2 contains an example for Linux platform.
- In the context of a particular platform, all units should be grouped by a framework. Framework in the context of this method is some tool or system with particular functions that could be called from external tools. Fig. 2 provides the SUBVERSION framework as an example.
- Each framework should be fulfilled with three main attributes:
  - ○Algorithms for actions of software configuration management. SUBVERSION framework provided in Fig. 2 contains algorithms for actions getRevisions() and refreshRevisions(). It means that the SUBVERSION framework could be used to implement actions getRevisions() and refreshRevisions() from the PIEM model.
  - ○Reusable functions, which could be called during using a particular framework. In other words, this is an interface that provides information 'What particular framework can do'. These functions should be parameterized: receive a set of parameters and return an expected result or error message. No values related to specific software development project should be hardcoded in the function body. In the context of Linux Shell scripts, all functions of SUBVERSION framework are stored in the script subversion_function.sh (Fig. 2).
  - ○Variables needed for implementation of a particular framework in a software configuration management server. They could be paths of tools needed for framework, environment variables, directories etc. For example, for the SUBVERSION framework variables could be paths of Subversion command line client, JAVA_HOME environment variable etc. In case of SUBVERSION framework, all such variables are stored in the script subversion_variables.sh (Fig. 2).

V. Use Case for Application of EAF Models and Methods

To illustrate practical application of provided models and methods, simplified software configuration management has been defined. There are two environments: DEV and TEST. Developers work with the DEV environment to develop new changes of some software based on JAVA technologies. Completed changes of the mentioned software should be saved in the Subversion repository, which is a system of version control and source code management in the described project. Software configuration management process should move changes from the DEV environment to the TEST environment. The scope of TEST environment is testing new functionality of software. Configuration manager has defined the following tasks of software configuration management to move changes between the mentioned environments:

- *getNewRevisions()*: detects ID of changes which are saved in the Subversion system, but are not delivered to the TEST environment.
- *prepareBaseline():* merges changes detected during a previous action from the DEV baseline to the TEST baseline.
- *makeBuild():* makes software build from the refreshed source code baseline of TEST environment.

- *installBuild():* installs the prepared build to the TEST environment.
- *updateRevisions():* marks particular revisions in the Subversion system as 'delivered to the TEST environment'.
- *sendNotification():* sends e-mails to testers of current project to inform that a new version of software is available from the TEST environment.

Software configuration management process with the mentioned actions should be reusable. Process should be implemented in Jenkins server by one job. Jenkins server is installed on Linux platform. The main task of application of EAF methodology is generation of a source code for the mentioned Jenkins job. Fig. 3 provides an overview of PIEM and CM models and connections between models and structure of views in Jenkins server.

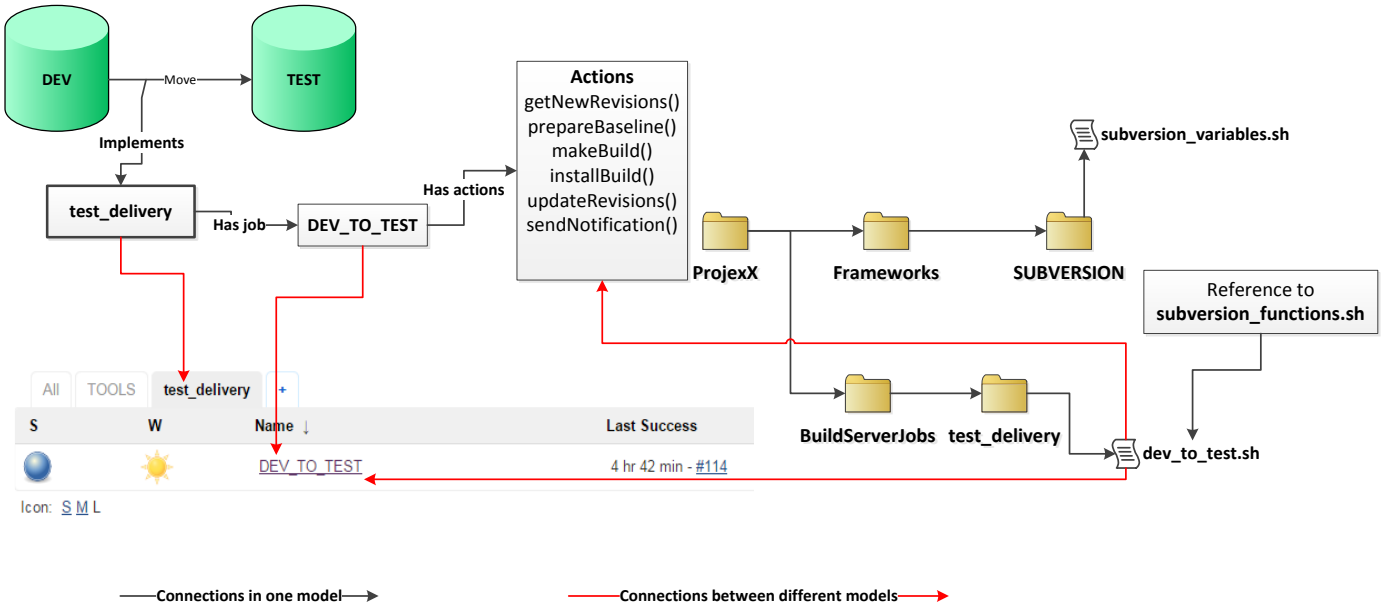

Fig. 3. Implementation of software configuration management by the EAF.

Fig. 3 demonstrates that the structure of views and jobs in Jenkins server is designed according to the structure of actions in the PIEM model. Jenkins server contains one view called "test_delivery" and one job called "DEV_TO_TEST". This job is implemented by a Code Model which contains script "dev_to_test.sh". Each execution of job "DEV_TO_TEST" will call the mentioned script. Script "dev_to_test.sh" contains algorithms for each action defined in the PIEM model and described before. Algorithms are selected by a configuration manager from the Solution Database. Code Model provided in Fig. 3 contains an example for the SUBVERSION framework, which implements the following actions: getNewRevisions(), prepareBaseline() and updateRevisions(). Other actions from the PIEM model are implemented by other frameworks and implementations are not represented in Fig. 3. Script "dev_to_test.sh" contains the reference to script with the Subversion reusable function (subversion_functions.sh) and the reference to script with Subversion variables (subversion_variables.sh). Mentioned references in script "dev_to_test.sh" are provided in Fig. 4.

```
15  #================SUBVERION=================================================================#
16  . ./Platforms/Linux_shell/ToolsFrameworks/SUBVERSION/subversion_functions.sh
17  . ./Projects/<ProjectX>/FrameworksVariables/SUBVERSION/subversion_common_variables.sh
18  export NODES_HOME="./Projects/<ProjectX>/NODES/TEST"
19
```

Fig. 4. References to the SUBVERSION framework.

Example of the SUBVERSION function "SVN_MERGE", which merges particular Subversion revisions from one source code branch to the other, is provided in Fig. 5.

```
# +=============================================================================================================================================+
#  Function to merge particular revisions from one SVN branch to other
#    This function is based on command: svn merge <URL> -c <rev1,rev2,revN>
#    After successfull merge changes of working copy will be commited.
#  Parameters:
#    $1 = SVN URL of remote branch
#    $2 = SVN user name
#    $3 = SVN password
#    $4 = SVN revisions from remote branch in format <1,4,5,6,...>
#    $5 = Directory of working copy where changes should be merged
#    $6 = Log message for commit result of merge
#    $7 = Option for resolving merge conflicts, for example: --accept mine-full or --accept theirs-full. Parametr is optional and could be empty
# +=============================================================================================================================================+
SVN_MERGE()
{
echo "[SVN_MERGE] Check count of parameters..."
if [ $# -lt 6 ]; then
    echo "[SVN_MERGE] This acript should be executed with 6 parameters at least!"
    exit 1
fi

############ Other body of function #####################################
```

Fig. 5. Example of the Subversion function.

Algorithms for actions from the PIEM model are also taken from the Solution Database. Fig. 6 provides a source code fragment of script “dev_to_test.sh” representing an algorithm for action prepare Baseline (). Algorithm uses two functions from the SUBVERSION framework: SVN_MERGE and SVN_COMMIT.

```
#Action prepareBaseline
#=============================================================================================================================================
#Goes througth all nodes of particular software
for nodes in `ls ${NODES_HOME}`
do
    #Inicializē konkrētas daļas mainīgus
    . ${NODES_HOME}/${nodes}
    #izpilda merge
    if [ "${SUBVERSION_MERGE_FLAG}" = "Y" ]; then
        if [ -f ${SUBVERSION_TEXT_FILE_FOR_REVISIONS} ]; then
            echo "Fails ar revīzijām: ${SUBVERSION_TEXT_FILE_FOR_REVISIONS}"
            local_revisions_for_merge=`cat ${SUBVERSION_TEXT_FILE_FOR_REVISIONS}`
            echo "Merge revisions ${local_revisions_for_merge} from ${SUBVERSION_URL} to working copy ${SUBVERSION_WORKING_COPY_DIRECTORY}
            SVN_MERGE ${SUBVERSION_URL} ${SUBVERSION_USER} ${SUBVERSION_PASSW} ${local_revisions_for_merge} ${SUBVERSION_WORKING_COPY_DIRE
            echo "Commit locals in working copy ${SUBVERSION_WORKING_COPY_DIRECTORY}"
            local_commit_log_message="Revisions ${local_revisions_for_merge} from ${SUBVERSION_URL}"
            SVN_COMMIT ${SUBVERSION_USER} ${SUBVERSION_PASSW} "${local_commit_log_message}" ${SUBVERSION_WORKING_COPY_DIRECTORY}
        else
            echo "Nodē ${GLOBAL_NODE_NAME} netika atrastas nepiegādātas revīzijas, merge nenotiks"
        fi
    else
        echo "Nav nepieciešams veikt merge darbību nodei ${GLOBAL_NODE_NAME}"
    fi
done
# END Action prepareBaseline
#=============================================================================================================================================
```

Fig. 6. Algorithm of action prepareBaseline() in script “dev_to_test.sh”.

Figs. 3–6 show that a source code for software configuration management process described in the current section is generated automatically by the EAF approach using the Solution Database, PIEM, PSAM and CM models. Only variables specific for a particular project should be defined in the Code Model to implement it in Jenkins server. No function or algorithm should be developed from scratch. The example described in the current section shows that the EAF approach with related models can help to decrease implementation time using existing tools, frameworks and executable source code units.

VI. Conclusion

The study provides a new model-driven approach for implementation of software configuration management. The main scope is to increase reuse of existing solutions and to reduce efforts to implement the process in other projects. Meta-models for the Platform Independent Environment Model, Platform Specific Action Model and Code Model are designed as an implementation example of the provided approach. In addition, a use case for designed models is given. Finally, differences from other approaches are underlined.

In order to continue research, it is necessary to carry out the following activities:

- With the help of experiment, to develop criteria that evaluate model benefits in software development projects;
- Based on the developed criteria, to evaluate benefits of designed models;
- To develop criteria in order to assess whether the developed model-driven approach for configuration management implementation corresponds to guidelines of ISO/IEC 15504, ITIL and CMMI standards;
- To design Code Models and transformation algorithms for other platforms;
- To add and improve tools and frameworks in existing platforms.

The approach provided in this article is abstract and only general stages, kinds of models and basic elements are defined. The authors hope that the new approach will generate new ideas because many useful lessons can be learned from different implementations of this model-driven approach.

ACKNOWLEDGEMENT

The research has been partly supported by the project eINTERASIA "ICT Transfer Concept for Adaptation, Dissemination and Local Exploitation of European Research Results in Central Asian Countries", grant agreement No. 600680 of the Seventh Framework Program Theme ICT–9.10.3: International Partnership Building and Support to Dialogues for Specific International Cooperation Actions – CP-SICA-INFSO.

REFERENCES

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

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

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

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

[5] Ragan, T., "21st-Century DevOps--an End to the 20th-Century Practice of Writing Static Build and Deploy Scripts," *Linux Journal*, vol. 2013, issue 230, pp. 116–120. Accessed on: Oct. 22, 2014.

[6] Azoff, M., DevOps: Advances in Release Management and Automation. [Online]. Available: http://electric-cloud.com/wp-content/uploads/2014/06/EC-IAR_Ovum-DevOps.pdf, 2014.

[7] Bartusevics, A. and Novickis, L., "Model-based Approach for Implementation of Software Configuration Management Process," in *MODELSWARD 2015: Proc. of the 3rd Int. Conf. on Model-Driven Engineering and Software Development*, France, Angers, Feb. 9–11, 2015. Lisbon: SciTePress, 2015, pp. 177–184. ISBN 978-989-758-083-3.

[8] Bartusevics, A., Novickis, L. and Lesovskis, A., "Model-Driven Software Configuration Management and Semantic Web in Applied Software Development," in *Recent Advances in Telecommunications, Informatics and Educational Technologies: Proc. of the 13th Int. Conf. on Telecommunications and Informatics*, TELE-INFO '14, Turkey, Istambul, Dec. 15–17, 2014. Istambul: WSEAS Press, 2014, pp. 108–116. ISBN 978-1-61804-262-0.

[9] Bartusevics, A. and Novickis, L. "Model-Driven Software Configuration Management and Environment Model," in *Recent Advances in Electrical and Electronic Engineering: Proc. of the 3rd Int. Conf. on Systems, Communications, Computers and Applications*, CSCCA"14, Italy, Florence, Nov. 22–24, 2014. Florence: WSEAS Press, 2014, pp. 132–140. ISBN 978-960-474-399-5, ISSN 1790-5117.

[10] Bartusevics, A., Novickis, L. and Bluemel, E., "Intellectual Model-Based Configuration Management Conception," *Applied Computer Systems*, vol. 15, 2014, pp. 22–27. ISSN 2255-8683. e-ISSN 2255-8691. http://dx.doi.org/10.2478/acss-2014-0003

[11] Bartusevics, A., Novickis, L. and Leye, S., "Implementation of Software Configuration Management Process by Models: Practical Experiments and Learned Lessons," *Applied Computer Systems*, vol. 16, 2014, pp. 26–32. ISSN 2255-8683. e-ISSN 2255-8691. http://dx.doi.org/10.1515/acss-2014-0010

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 Bachelor (2008) and Master (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 the Head of the Division of Software Engineering. He obtained *Dr.sc.ing.* degree in 1980 and *Dr. habil. sc. ing.* degree in 1990 from the Latvian Academy of Sciences. 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, FP7. He is a corresponding member of the Latvian Academy of Sciences. His research fields include applied software system development, business process modelling, eLogistics, international cooperation, web-based applications. He is the coordinator of FP7 eINTERASIA project.
E-mail: lnovickis@gmail.com

Stefan Leye holds a Diploma in Mechanical Engineering (2011). Primary field of study – integrated product development. Currently he works at the Fraunhofer Institute for Factory Operation and Automation IFF, Magdeburg – Germany, as a Project Manager at 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