ALM Tool Data Usage in Software Process Metamodeling

Petr Pícha, Přemek Brada
Department of Computer Science and Engineering & New Technologies for the Information Society Research Center
University of West Bohemia
Pilsen, Czech Republic
{ppicha, brada}@kiv.zcu.cz

***Abstract*—Project Management and Software Process Improvement (SPI) are essential parts of software engineering. A multitude of project management techniques and tools, such as Application Lifecycle Management (ALM) ones, are available, and there is an abundance of software methodologies, process metamodels and best practice descriptions. Despite the role of tools in enacting the processes, the underlying domain models used in these two fields – ALM tools and software process metamodels – often significantly differ. This hinders the project execution analysis using the data available from the tools, such as the verification of the project's alignment with a given methodology or the detection of bad practices (anti-patterns). In this paper we describe our work towards enabling such analyses. First we survey the domain models of major process metamodels and tools, and discuss the commonalities and open issues. We then propose a software process metamodel inspired by some of the established metamodels and reflecting the structures of data which can be mined from project management tools. The proposed metamodel is being validated by a prototype process data repository with import interfaces for major ALM tools.**

Keywords—ALM; software process; metamodel; project management

I. Introduction

In the last several decades Software Engineering made significant progress in terms of providing ever better tools and guidance for software development process. This includes a plethora of methodologies, best practices (patterns) and supporting tools for process modeling and Application Lifecycle Management (ALM). These are then incorporated into a particular software process, used to tailor it for immediate purpose, monitor and provide data for Project Management (PM). That gives PM the knowledge necessary to decide either to locally adjust a course of action of a project or apply Software Process Improvement (SPI) practices, leading to more effective changes. Yet, despite the volume of guidance materials, some project management bad practices (anti-patterns) still occur on regular basis indicating there is a room for improvement.

Our area of interest is the detection of such anti-patterns, using the project data stored in the ALM tools, which reflect the day-to-day reality of a project. Examples include over- and underestimation of tasks, an iteration of development not ending with a stable release, straying from a declared methodology, or a person performing tasks inconsistent with their role in the project. An enabling factor for such detection of process (anti-)patterns is their unified representation and identification in the ALM data. However, efforts to create a common representation suitable for the data analysis are scarce if any, leaving the ALM data potentially unused.

In this paper we present a process metamodel based on ALM data as a first step in an effort to create an analytical tool to help PM staff identify, avoid and mitigate these anti-patterns. We call this tool Software Process Anti-patterns Detector (SPADe). Section 2 explains the motivation for the SPADe approach as a whole, the need to start with the structure for storing and analyzing the data and the decision for it to have a form of a metamodel. Section 3 discusses some important related work. Sections 4 and 5 describe our approach and design in creating the metamodel. The metamodel itself, and its main contributions over other models are described in section 6, and we conclude the paper with hinting on the future work.

II. Motivation

Though most software development projects use ALM tools, which combine the supporting tools such as Version Control Systems (VCS) and issue tracking applications into a coherent whole, the structure of the data they provide is heavily dependent on the given tool and the data itself are largely unused for any deeper analysis of the current state of the project or anti-pattern identification. To enable such interesting analyses, a system is needed that would collect data from various ALM tools, analyze them, identify anti-patterns (as potential threats to the project success) and compare the current project with other past projects of similar size, scope and methodology. The user-facing tool would then present the findings in a dashboard-typed UI with warnings, hints and projections and would be used mostly by project managers and development team leaders. Fig. 1 shows an architectural model of such a system.

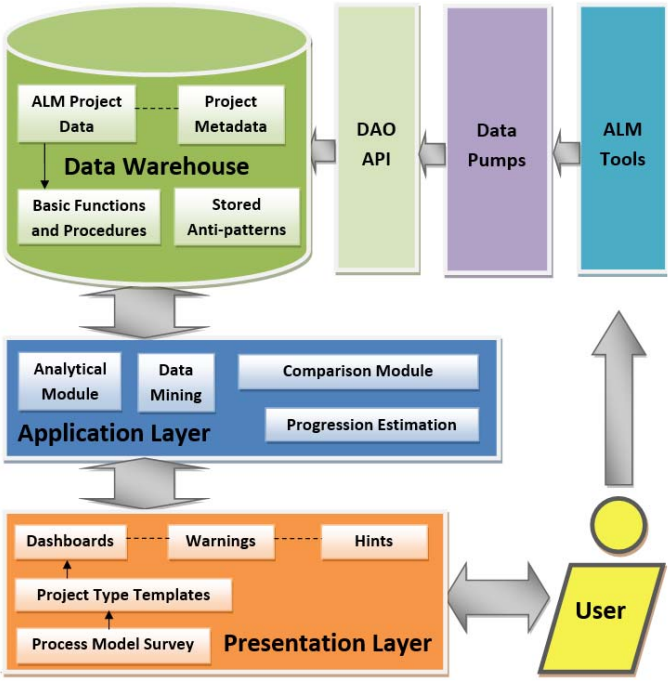

Figure 1. SPADe architecture

The first logical step in creating such a tool is designing its mechanism of collecting and storing both current and past project data. To this end, we need a unified model for the data collected from the analyzed projects. The challenge in this respect is that the projects are often based on different methodologies and use different ALM tools. Also, the model should be able to represent the identifiable anti-patterns in the project data.

As described in [1], there are many approaches to software process modelling. But the review shows the preference of model-based approach (metamodels) rising steadily throughout the practice since 2006. Since model-based approach also supports object-oriented view and a domain metamodel is easily convertible to ERA model for the database, it forms the base of the approach we use as presented in this paper.

III. Related Work

A literature review from 2013 focusing on software process language modeling by García-Borgoñón, et al. [1], covers model-, grammar- and UML-based approaches to capturing different methodologies and processes. Apart from OMG Software & Systems Process Engineering Meta-Model (SPEM) 2.0 [2], probably the most well-known process metamodel, it included some of its extensions, such as Enactable SPEM (eSPEM) [3] and Variability SPEM (vSPEM) [4][5]. It also contains the standardized metamodel of ISO 24744 [6][7] and its derivate PMMM [8]. Another standardized model, ISO 29110 [9][10] is focused on processes for very small entities of up to 25 people. SPEM 2.0 is also used in IBM Rational Method Composer (RMC) [11], a tool for modeling and tailoring processes, the products of which can be exported and used as a project templates for IBM Rational Team Concert (RTC), an ALM tool. Another important resource is Open Service for Lifecycle Collaboration (OSLC) and its specifications, mainly for Change [12] a Configuration [13] Management.

Furthermore, some work on data analysis gathered from development environment including VCS tools is being done by Grambow, Oberhauser, and Reichert [14][15] and used to adjust personal software process of a developer in real time. Joblin, et al. [16][17] try to find patterns and anti-patterns in social interactions of the project staff. In papers by Settas, Stamelos, et al. [18][19], researchers aim to identify anti-patterns in collaboration and communication based on personality assessment. Ramsin, et al. study the process patterns in different fields [20][21] and their automated structuring and analysis in an attempt to establish a basic set of high-level process patterns to set up a generic framework. Other authors present a procedure for extracting process patterns from any software development methodology [22]

In none of the sources analyzed as related work did we find a process metamodel that would use the ALM data in unified form to reflect the day to day reality of project execution or to look for anti-patterns. Most of them are either high-level models with insufficient level of detail and without much variety in entity classes (e.g. SPEM), or are heavily tailored for their respective proprietary or domain purposes inconsistent with our goals. Therefore a need arose to design our own ALM-data-based software process metamodel.

IV. Approach and Design

Before describing the metamodel itself, we need to clarify the scope of our effort:

- Which ALM tools will we consider to mine data from?
- How to come up with a metamodel that fits our selected subset of tools?
- Which methodologies or processes should the metamodel be able to capture?
- How will it map to other metamodels?
- Which technologies should we use to implement it?

In answering the first question, we first need to consider that not all ALM tools are equal. Tools used for project monitoring purposes can most easily be divided to issue tracking, Version Control Systems (VCS) and full-fledged ALM tools. Therefore, in common practice development teams frequently use several tools to manage a single project.

For our analysis, we wanted to consider a reasonably large set of most commonly used tools with an API for data mining. The resulting set consists of Git and Apache Subversion (SVN) as VCS tool representatives (widely used and interfacing with issue tracking tools), and Atlassian Jira, Bugzilla, Redmine (all three widely used according to Eclipse Survey [23]), Assembla (highly accessible cloud application), IBM Rational Team Concert (RTC; due to its direct relationship to RMC, SPEM and Rational Unified Process – RUP) and GitHub (for its recent massive popularity). Other tools have also been considered and could be added to the set in the future. The set

consists of such tools which are commonly available, have suitable API and are used for real life ALM purposes in our proximity (meaning either at our university or by one of our industry partners).

To address the second question, we decided to start from the intersection of the tool domain models, plus we added the data appearing in most (but not all) of them with the potential of being helpful to future analyses. For example, aside from estimated and spent time, most issue tracking tools also allow for the *remaining estimate* attribute in tasks which can be valuable in some anti-pattern detection (e.g. estimation accuracy is not improving enough or gets even worse after re-estimation). The OSLC specifications [12][13] have also been used to draw concepts and terminology. Table 1 shows, for the fragment of the above set of tools, the mapping between some of their issue tracking data, OSLC concepts and the SPADe metamodel. The intersection approach also gives us a certain latitude in selecting the initial ALM tools set while maintaining the usability of the metamodel for other tools in the future because it makes use only of the common concepts and omits most of the idiosyncratic capabilities of any particular tool. Therefore inclusion of any particular tool into the set has less impact on the metamodel as the set grows in size.

As for the methodologies or particular processes the metamodel should be able to represent, we focused primarily on sequential and iterative approaches to development. The reason is that the union of these gets us the most comprehensive project modeling and segmentation scheme, covering projects with phases (with or without milestones), iterations, activities (sets of related tasks with unified goal), and individual tasks. This structure is capable of modeling even other major families of methodologies, such as ad-hoc (going from project straight to tasks), agile and lean (by omitting phases), or hybrid ones, like Disciplined Agile Delivery (DAD) [24].

Table 2 illustrates the occurrence of phases milestones and iterations in some major methodologies (projects, activities and tasks are present in all of them). As the differences between methodologies of the same approach (e.g. Waterfall vs. V-Model, RUP vs. OpenUP, etc.) are mostly in the practices used and not in partitioning the lifecycle, this paradigm should provide enough support for general process modeling in SPADe. Furthermore, smaller lifecycle fragments used in processes are strictly date-related (weeks or days). In ALM tools these are represented solely by start and due dates of tasks, so there is no need to incorporate them in the model.

Other major concepts in software process modeling like roles and artifacts are methodology independent (meaning they appear in all of them), as are concepts appearing in ALM tools themselves, like software configuration, artifact change or development branch. In summary, we aimed the design of the metamodel with the intent to cover all major methodologies.

Neither the authors of [1] nor we have come across a metamodel directly usable for our purposes. Therefore the actual metamodel draws mostly from SPEM 2.0 [2] as it is the most commonly known and extended upon process metamodel.

TABLE I. MAPPING OF ENTITIES BETWEEN ALM TOOLS AND SPADE

ALM Tools Objects				**OSLC Concepts / Terminilogy**	**SPADe Entities/(Attributes)/Relations**
Jira	***Bugzilla***	***Redmine***	***Assembla***		
Issue	Issue	Issue	Ticket	Defect, Enhancement, ReviewTask, Task, ChangeRequest	Work Unit
User	User	User	User	Person	Identity, Person
Project	BugzillaProject	Project	Space	Container	Project
Attachment	Attachment	Attachment	Document	Attachment	Artifact
Comment	Comment	Journal	Ticket Comment	Comment	Change (comment)
Component	BugzillaComponent Classification	IssueCategory	-	-	Work Unit (category)
Version	BugzillaVersion	Version	Milestone	Container	Iteration/Phase
ChangeHistory	-	Journal	Ticket Comment	Change	Configuration – Change – Artifact
ChangeItemBean	-	Journal Details	Ticket Comment	Change Set	
Worklog	-	Time Entry	Ticket Comment	-	-
ProjectCategory	-	Project	Group(type=”space”)	-	DevelopmentProgram
GroupBean	-	Group	Group(type=”user”)	User/Work Group	Group
IssueLink	IssueLink	IssueRelation	Ticket Association	Change Request Relationship Properties	Work Unit – Work Unit
Issuelink Type	-	-	-		
Priority	BugzillaPriority	IssuePriority	Custom Field (name=”priority”)	Priority	Work Unit (priority)
Status	BugzillaStatus	IssueStatus	Ticket Status	State	Work Unit (status)
IssueType	BugzillaIssueType	Tracker	Custom Field (name=”type”)	Defect, Enhancement, ReviewTask, Task, ChangeRequest	Work Unit (type)

TABLE II. CONCEPT OCCURRENCE IN METHODOLOGIES

Methodology	Concept		
	Phases	***Milestones***	***Iterations***
Ad-hoc	No	No	No
Waterfall	Yes	No	No
V-Model	Yes	No	No
Unified Process	Yes	Yes	No
Scrum	No	No	Yes (Sprints)
Kanban	No	No	Yes
DAD	Yes	Yes	Yes

However SPEM does not have entities to represent many of theday to day realities of the projects stored in ALM data, such as a configuration, development branch or a person. Therefore we had to expand it to a significant degree.

Also, there is no need for our metamodel to differentiate between base entities (*Role*, *Task* and *Work Product*) and their respective *Use* concepts as SPEM does because ALM project data (tickets, issues, revisions, etc.) represent the actual entity usage in the project execution. From ALM-tool-based point of view roles are mostly static, meaning their definition and usage hold effectively the same information throughout the project, and the artifact states (the main feature of Work Product Use entities in SPEM) are represented through their changes over time. We also have no use for the *Guidance* element in SPEM, as it represents various checklists, tool manuals, examples and other supporting materials, which in ALM tools could be used as an attachment and – from our perspective – are viewed as artifacts.

The OSLC specifications could not be used in their entirety because some of the tools in our set do not implement them and some of the specification have not yet reached a definitive stable version [13]. Nevertheless, the specifications were used to check for additional potentially useful concepts and terminology.

Some terminology has also been adopted from similarly focused Software Engineering Workflow Language (SEWL) [15] and RUP methodology [25]. In future work on the SPADe approach, vSPEM [4][5] can be used in future work for model variability in different methodologies.

Finally, concerning the question of implementation technology, the goal for the SPADe tool is to be as platform independent as possible. In practice it mainly needs to consider the APIs of the ALM tools which are used for transferring the data into the unified repository through data pumps (import interfaces).

V. THE SPADE METAMODEL

The diagram of our metamodel which is the result of the analysis and considerations discussed above is depicted in Fig. 2. In the subsequent parts we describe its less obvious aspects, discuss the current status and the envisaged usages of the metamodel.

A. Metamodel Elements and Structures

The black elements represent core entities to be mostly mapped from ALM data directly. Blue elements depict concepts that can be identified only using some degree of analysis or user participation (e.g. phases in iterative development usually do not have direct representation in ALM tools). Red elements represent entities beyond single project data (i.e. at the level of *Development Program*) or concepts on a higher level of abstraction, for which an outside definition is needed (i.e. *Activity*, *Criterion* and *Competency*). Grey associations symbolize inheritance/generalization and entities with names in italics are abstract.

All entities have an identifier (*ID*) and *external ID* (the technical ID as employed by the source ALM tool installation). Almost all the entities also have name, with the exceptions of *Change*, and description (apart from *Branch*, *Configuration* and *Person*) attributes.

The majority of the metamodel concepts share the definition with SPEM [2] and RUP methodology [25] as discussed above or are a straightforward result of Table 1. Some entities and attributes, however, deserve further clarification:

- *Artifact* – configuration item; the term has been adopted from RUP (in SPEM it is used for one kind of *Work Product*);
    - *type* – in the realm of ALM data it can be a file or a folder in the VCS repository, an uploaded file or a wiki page in the tool (in the future also e.g. emails);
- *Authored Entity* – superclass extended by entities for which author and timestamp of creation is stored (*Work Item* and *Configuration*);
    - *created* – timestamp of creation;
- *Branch* – development branch from VCS;
    - *type* – trunk / master branch or other;
- *Change* – represents repository commit, file upload, ticket (*Work Unit*) or wiki page edit;
    - *type* – addition, deletion or modification of an artifact;
- *Configuration* – apart from VCS revisions also any state resulting from any *Change*;
    - *number* – revision identifier in VCS user interface (not necessarily the same as *external ID*);
    - *type* – revision in VCS or other state (e.g. after editing wiki page or ticket);
    - *tags* – list of names of tags associated with the configuration in VCS, if it is a revision and there are any;

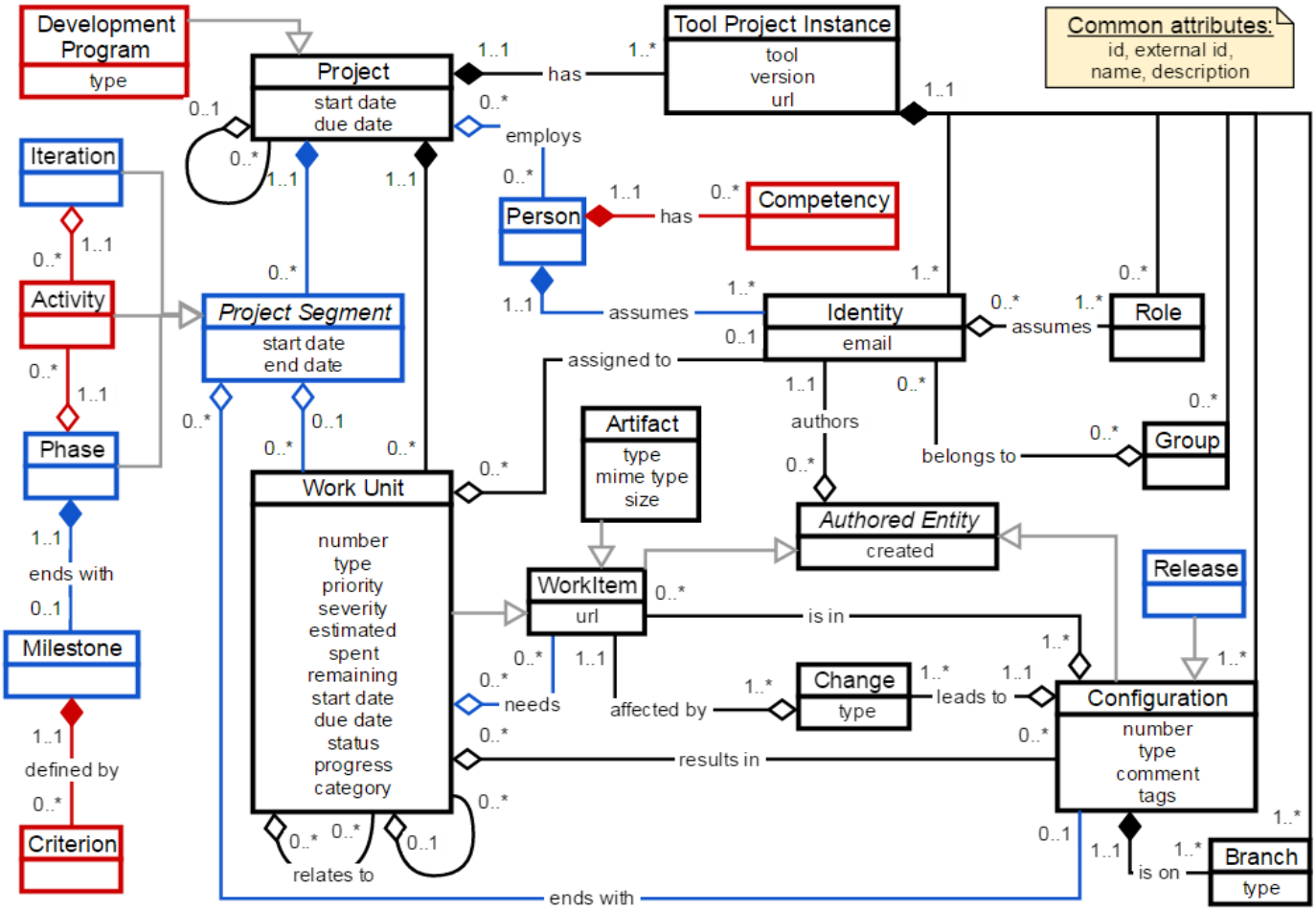

Figure 2. SPADe domain metamodel

- *Competency* – represents expertise, certification, education or any such asset a person has;
- *Criterion* – a partial goal of a phase, an indicator of achieving its milestone;
- *Development Program* – a group of related projects (stored as a parent project), e.g. series of products, application versions for different platforms, etc.;
- *Identity* – represents user account in a particular tool, since a project can use more than one tool for ALM purposes (as discussed above);
  - *name* – represents login or username;
  - *description* – represents full name filled in by a user in the ALM tool registration form – grounds for matching identities to a *Person* instance;
- *Person* – the consolidation of identities on this entity allows for statistics of a particular person throughout various projects and tools used to be calculated;
- *Release* – VCS tag or other important configuration;
- *Tool Project Instance* – holds information about a particular ALM tool installation, from which the project data were obtained;
  - *url* – source link of the project data;
- *Work Item* – superclass for both *Artifact* and *Work Unit*;
  - *url* – direct link to the actual work item (in repository, wiki, ALM tool GUI, etc.);
- *Work Unit* – task; the term has been adopted from SEWL (e.g. [15]) because it most accurately describes the use of the concept in our context;
  - *number* – ticket identifier in issue tracking tool (unique inside a project);
  - *type* – bug, task, feature, enhancement, etc.;
  - *estimated*, *spent* – time;
  - *remaining* – estimated remaining time;
  - *status* – issue lifecycle status (new, assigned, resolved, closed, invalid, etc.);
  - *progress* – percentage of the task done.

- The relation between *Work Unit* and *Work Item* is blue, because unless the users use a specific capability of the ALM tool (attach a file or use link in a task description, specify a particular relation between two tasks, etc.), there is no way to directly assign a prerequisite to a task. Therefore the information must be mostly read from the context of the task.
- The self-referencing associations on *Project* and *Work Unit* represent that they have subprojects or subunits respectively.
- Other non-labeled associations represent the "contains" relationship.
- There is no relation between *Phase* and *Iteration* – because a milestone achievement (end of a phase) does not have to correspond with the end of an iteration.

The rest of the metamodel is mostly a way to capture common concepts (e.g. *Milestone*, *Branch*, *Configuration*, *Change*, etc.) and represent metadata needed for accurate analysis (e.g. *Tool Project Instance*).

The metamodel is implemented in the data layer of the SPADe tool. As most of the selected ALM tools have Java or REST APIs, Java was selected as the best choice of technology for the entire implementation. It also fulfills the requirement of platform independency. Furthermore, by using a Java Persistence API (JPA) implementation based on Data Access Object (DAO) interfaces the application will be also independent of the underlying data warehouse technology.

B. Evaluation and Current Status

Because the metamodel is based on the intersection of available ALM data, the mapping of another ALM tools into the SPADe structures is only a question of implementing the corresponding data pumps compatible with the data warehouse input API. Should the data structures of any ALM tool be insufficient for the analysis, the very ALM nature of the tool is in question and most probably falls out of the scope of our efforts.

The presented SPADe metamodel was specifically designed to be able to capture processes based on most major methodologies on the level of detail of our interest (i.e. the level extractable from ALM tools data). Table 3 shows the mapping between our metamodel and the much more detailed and theory-based SPEM. Although SPEM could by itself be seen as a suitable model for our purpose, we found it did not correspond well to our needs. First, its vast number of concepts and entities are outside the scope of our effort (e.g. *Guidance*, separate *Definition* and *Use* entities, *Steps*, *Additional Performers*, *Work Product State*, etc.). Second, other SPEM entities are not specific enough or do not have needed attributes for modeling some ALM concepts (e.g. commit, configuration, branch, etc.) as discussed in section 4. Another reason why SPEM is not suitable for our purposes is that it does not feature entities for metadata elements (e.g. *Tool Project Instance*), actual development team members (i.e. *Person*) and related projects collection (i.e. *Development Program*).

TABLE III. SPEM TO SPADE MAPPING

SPEM 2.0 Concept	SPADe Entity (Attribute)
Activity	Activity
Category	Work Unit (type, category), Artifact (type), Configuration (type)
Iteration	Iteration
Milestone	Milestone
Phase	Phase
Process	Project
Role Definition / Use	Role
Task Definition / Use	Work Unit
Work Product Definition / Use	Artifact, Configuration, Release

The current version of the metamodel has been implemented and its inner consistency tested via a JPA implementation (namely Hibernate) and a MySQL database. The input DAO interface has been implemented as well. The metamodel has been validated against the majority of the selected ALM tools and major methodologies described earlier. Fig. 3 shows a fraction of a student project modeled by SPADe entity instances (only some attributes are shown).

At the moment we are finalizing the validation against GitHub which has been added to the set only recently. Nevertheless we do not expect this to result in major changes due the "intersection-driven" model design.

C. Target Usage

As a process metamodel which is by design aware of the data which can be obtained from project execution in ALM tools, the SPADe metamodel can be used for a variety of different purposes.

To support the analysis of process data and the detection of anti-patterns, a concrete process model conforming to SPADe metamodel can be used as a template for projects using the given methodology, which they should stick to.

Straying from the template can be classified as an identified anti-pattern. This can be detected in several ways: (1) the non-existence of any ticket in the ALM data representing a task (modeled by a *Work Unit* instance) in the template based on a value of one or more attributes (such analysis will probably require tickets natural language processing), (2) an absence or surplus of some instances of the metamodel entities or their attributes (e.g. too few iterations, commit without a comment) or their relationships (e.g. no release at the end of iteration).

Anti-patterns can also be detected as aggregated attribute values on the modeled process data (e.g. difference between estimated and spent time across all work units), or span over several entity instances and their relations (e.g. analytical task performed by a person with insufficiently competent role). On a higher level anti-patterns can be more complex and methodology or scope dependent. These can include cases such as iterations being too long, too much time spent on iteration planning, poor user-level feature time estimates very late in the project, or user bug reports not being handled in a reasonably

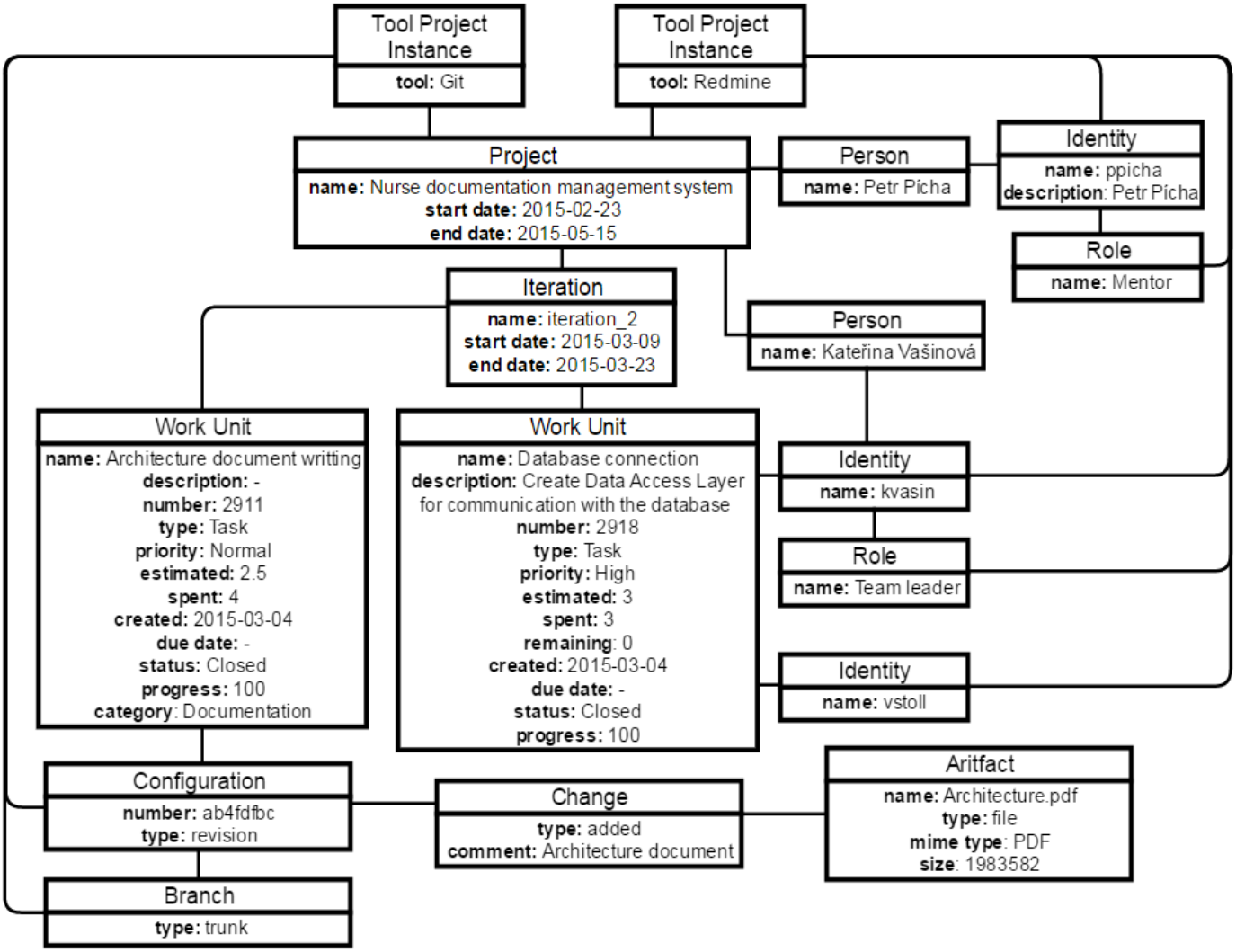

Figure 3. Partial project structure modeled with SPADe metamodel

short timeframe. Such occurrences can indicate either (1) the process is not iterative despite being declared as such, or (2) testing will be suppressed at the end of the project in order to fix the fundamental functionality, which has its known undesirable consequences.

The anti-patterns themselves can also be modeled via the SPADe metamodel and then identified in a reverse fashion, meaning "the submodel whose occurrence signals the anti-pattern".

Because we use the intersection of ALM tools domain models the application using the metamodel also has the opportunity to compare data from various projects independent of their tool of origin. This is primarily intended to be used for prediction of future progress of a project, given the current course and based on past projects of similar scope and methodology. This will allow the application to warn the user in a fashion like "if you do not do A, you will most likely miss your due date by X", or "similar projects with the same identified anti-patterns failed in X percent of the time". In a similar fashion, the same can be done on the level of *Development Program* instances.

The data in the warehouse, structured according to the SPADe metamodel, can of course be used by other applications focusing on different analysis than anti-pattern detection, opening a multitude of possibilities for future usage. The metamodel can for example provide data for statistics about a particular person, or more accurately, about their activity and behavior throughout many projects using different ALM tools. Such data can be subsequently used in further research.

VI. Conclusions

We have proposed the SPADe metamodel, a new software process metamodel geared to the utilization of ALM tool data and based on well-established practices and other metamodels. Our metamodel uses its unified structure to represent and store real project data from various ALM tools and can be used for projects following various methodologies and processes. It is designed to enable project data analysis, namely to represent and detect process anti-patterns with the intended use in a tool

helping project managers and development team leaders with project management and software process improvement activities. As there is a lack of comparable unified and domain specific metamodels we see a wide range of its potential contributions in the field of software engineering.

Our primary focus for future work is a thorough validation of the SPADe metamodel, performing a detailed analysis of the data mineable from ALM tools and practically by fully implementing the data warehouse and data pumps for selected ALM tools.

In the near future we also want to create the methodology templates, collect a comprehensive set of detectable anti-patterns and establish their appropriate classification and representations.

ACKNOWLEDGMENT

The work was supported by the UWB grant SGS-2013-029 Advanced Computer and Information Systems.

REFERENCES

[1] L. García-Borgoñón, M. A. Barcelona, J. A. García-García, M. Alba, and M. J. Escalona, "Software process modeling languages: A systematic literature review," Information and Software Technology, vol. 56(2), Feb. 2014, pp. 103-116, doi:10.1016/j.infsof.2013.10.001.

[2] OMG Group, Software & Systems Process Engineering Meta-Model Specification, OMG Std., 2008.

[3] R. Ellner, S. Al-Hilank, J. Drexler, M. Jung, D. Kips, and M. Philippsen, "eSPEM–A SPEM extension for enactable behavior modeling," in Modelling Foundations and Applications, Springer Berlin Heidelberg, 2010, pp. 116-131.

[4] T. Martinez-Ruiz, F. García, M. Piattini, and J. Münch, "Applying AOSE concepts to model crosscutting variability in variant-rich processes," 2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), IEEE, Sep. 2011, pp. 334-338, doi:10.1109/SEAA.2011.58.

[5] T. Martinez-Ruiz, F. García, M. Piattini, and J. Münch, "Modelling software process variability: an empirical study," IET Software, vol. 5(2), Apr. 2011, pp. 172-187, doi:10.1049/iet-sen.2010.0020.

[6] F. B. Ruy, R. A. Falbo, M. P. Barcellos, and G. Guizzardi, "An ontological analysis of the ISO/IEC 24744 metamodel," Proc. Eighth Int. Cpnf. Formal Ontology in Information (FOIS 2014), IOS Press, Sep. 2014, pp. 330-343, doi:10.3233/978-1-61499-438-1-330.

[7] C. Gonzalez-Perez, "Supporting situational method engineering with ISO/IEC 24744 and the work product pool approach," Situational Method Engineering: Fundamentals and Experiences, vol. 244, Sep. 2007, pp. 7-18, doi: 10.1007/978-0-387-73947-2_3.

[8] S. Jablonski, B. Volz, and S. Dornstauder, "A meta modeling framework for domain specific process management," 32nd Annual IEEE International Computer Software and Applications (COMPSAC'08), IEEE, Jul. 2008, pp. 1011-1016, doi: 10.1109/COMPSAC.2008.58.

[9] R. V. O'Connor and C. Y. Laporte, "Software project management in very small entities with ISO/IEC 29110," Systems, Software and Services Process Improvement, vol. 301, Jun. 2015, pp. 330-341, doi 10.1007/978-3-642-31199-4_29:.

[10] R. V. O'Connor and C. Y. Laporte, "Using ISO/IEC 29110 to harness process improvement in very small entities," Systems, Software and Services Process Improvement, vol. 172, Jun. 2011, pp. 225-235, doi: 10.1007/978-3-642-22206-1_20.

[11] P. Kroll, Introducing IBM Rational Method Composer, The Rational Edge, 2005.

[12] "Open Services for Lifecycle Collaboration Change Management Specification Version 3.0", Open Services for Lifecycle Collaboration, 5 February 2014.

[13] "Open Services for Lifecycle Collaboration Configuration Management 1.0", OASIS Editor's Draft, Open Services for Lifecycle Collaboration, 21 April 2016.

[14] G. Grambow, R. Oberhauser, and M. Reichert, "Towards a workflow language for software engineering," 10th Int. Conf. on Software Engineering (SE'11), IEEE, Feb. 2011, doi:10.2316/P.2011.720-020.

[15] G. Grambow, R. Oberhauser, and M. Reichert, "Towards automated process assessment in software engineering," 7th Int. Conf. on Software Engineering Advances (ICSEA'12), Nov. 2012, pp. 289-295.

[16] M. Joblin, S. Apel, and W. Mauerer, "Evolutionary trends of developer coordination: A network approach," unpublished.

[17] M. Joblin, W. Mauerer, S. Apel, J. Siegmind, and D. Riehle, "From developer networks to verified communities: a fine-grained approach," Proc. 37th International Conference on Software Engineering (ICSE'15), IEEE Press, May 2015, pp. 563-573.

[18] D. Settas and I. Stamelos, "Towards a dynamic ontology based software project management antipattern intelligent system," 19th IEEE Int. Conf. Tools with Artificial Intelligence (ICTAI 2007), IEEE, Oct. 2007, pp. 186-193, doi:10.1109/ICTAI.2007.43.

[19] D. Settas and I. Stamelos, "Using ontologies to represent software project management antipatterns," 19th Int. Conf. Software Engineering & Knowledge Engineering (SEKE 2007), Knowledge Systems Institute Graduate School, Jul. 2007, pp. 604-609.

[20] M. Khaari and R. Ramsin, "Process patterns for aspect-oriented software development," 17th IEEE Int. Conf. and Workshop on Engineering of Computer Based Systems (ECBS), IEEE, Mar. 2010, pp. 241-250, doi: 10.1109/ECBS.2010.33.

[21] E. Kouroshfar, H. Y. Shahir, and R. Ramsin, "Process patterns for component-based software development," Component-Based Software Engineering, vol. 5582, 2009, pp. 54-68, doi: 10.1007/978-3-642-02414-6_4.

[22] M.F. Gholami, P. Jamshidi, F. Shams, "A procedure for extracting software development process patterns," 4th UKSim European Symp. Computer Modeling and Simulation (EMS), IEEE, Nov. 2010, pp. 75-83, doi: 10.1109/EMS.2010.24.

[23] "The open source developer report: 2011 Eclipse community survey," The Eclipse Foundation, 2011.

[24] S. W. Ambler and M. Lines, Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise, IBM Press, 2012.

[25] P. Kroll, and P. Kruchten, The rational unified process made easy: A practitioner's guide to the RUP, Addison-Wesley Professional, 2003.