Towards Architect's Activity Detection Through a Common Model for Project Pattern Analysis

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

Ralf Ramsauer[1], Wolfgang Mauerer[1, 2]
[1]Laboratory for Digitalisation, OTH Regensburg
[2]Siemens AG Munich
Germany
{ralf.ramsauer, wolfgang.mauerer}@oth-regensburg.de

***Abstract*—Software development projects leave a large amount of data in repositories of Application Lifecycle Management (ALM) tools. These data contain detailed histories of their respective projects, their results and decisions made along the way. Analysis of such data helps uncover various interesting facts about projects, e.g. their socio-technical structures and the actual (vs. purported) roles of team members. Based on experiences with tools supporting our research we are convinced that it is feasible to consolidate data from different ALM tools, tapping into the situation common in real-life projects. In this paper we report on our work towards a shared common data model and tool integration aimed at improved project analysis. We discuss how this can help in the identification of architects in the project organizational structures, their activity patterns and collaboration with other team roles.**

Keywords—empirical software engineering; project patterns; ALM data; software architect; activity analysis

I. Introduction

Empirical software engineering is based on data on technical, process and societal structures of actual projects. While in-depth analyses of such data (see, e.g., [1]) often discover valuable and unexpected facts, the analysis is often hindered by the fact that different projects (even in the same company or open source community) use different Application Lifecycle Management (ALM) tools. Obtaining coherent data is often a difficult process performed by researchers with the help of ad-hoc tools. Existing interoperability standards such as XMI or OSLC[1] are insufficient and therefore inapplicable to such research efforts.

The analysis of software projects using ALM data is a common interest of both author research teams (UWB Pilsen, OTH Regensburg). In this paper, we describe a common data model that consistently captures the content of various ALM tools. We then describe the two tools currently under development at OTH and UWB respectively, Codeface and SPADe (Software Process Anti-patterns Detector). The common model implementation is used to obtain and share data from several version control and issue-tracking systems between the tools, and lead to enriched information for analyses done by both research groups. Finally, we discuss the practical integration of the tools resulting from a joint work by the authors of this paper, and its benefits.

Our work differs from the contributions of Draheim and Pekacki [2] in using both Version Control Systems (VCS) and issue-tracking tools, and from German [3] and Grambow, Oberhauser and Reichert [4] by being non-specific to one set of tools. Its results could be utilized by other researchers, for example for personal workflow modifications [5][6].

II. ALM Data and Common Model

Any quantitative analysis of software development projects requires accurate data that describes the various aspects of their history and product construction. Because any larger software development project needs appropriate tool support to track status, activities and artefacts created by project members, data from these tools are ideally suited to perform such analyses. ALM tools, which integrate VCS and issue-tracking tools, are a commonly used family of tools and (if used properly) offer an objective and detailed history of collaboration patterns and activities of individual team members.

Different tools vary strongly in their data models, the availability of analytical and statistical modules, or the granularity of data stored (e.g., rigorous RTC[2] vs. lightweight GitHub[3]). However, there is a common underlying set of entities that all ALM tools use, and that data structures can thus be mapped be mapped to these entities. In previous work by some of the authors [7], we have devised such a common data model (Fig. 1) by studying the data structures of several ALM tools. It can reflect major software development process methodologies (among others, see [8] for details, ISO 24744 and SPEM – the Software and Systems Process Engineering Meta-Model), enabling process-centric analysis.

Since not all ALM tools store the project data at the same level of detail, some meta-model entities need to be analytically inferred. In Fig. 1, the black entities can be (more-or-less directly) obtained from ALM data – among them are names

[1] Open Services for Lifecycle Collaboration, https://open-services.net/

[2] http://www-03.ibm.com/software/products/en/rtc

[3] https://github.com/

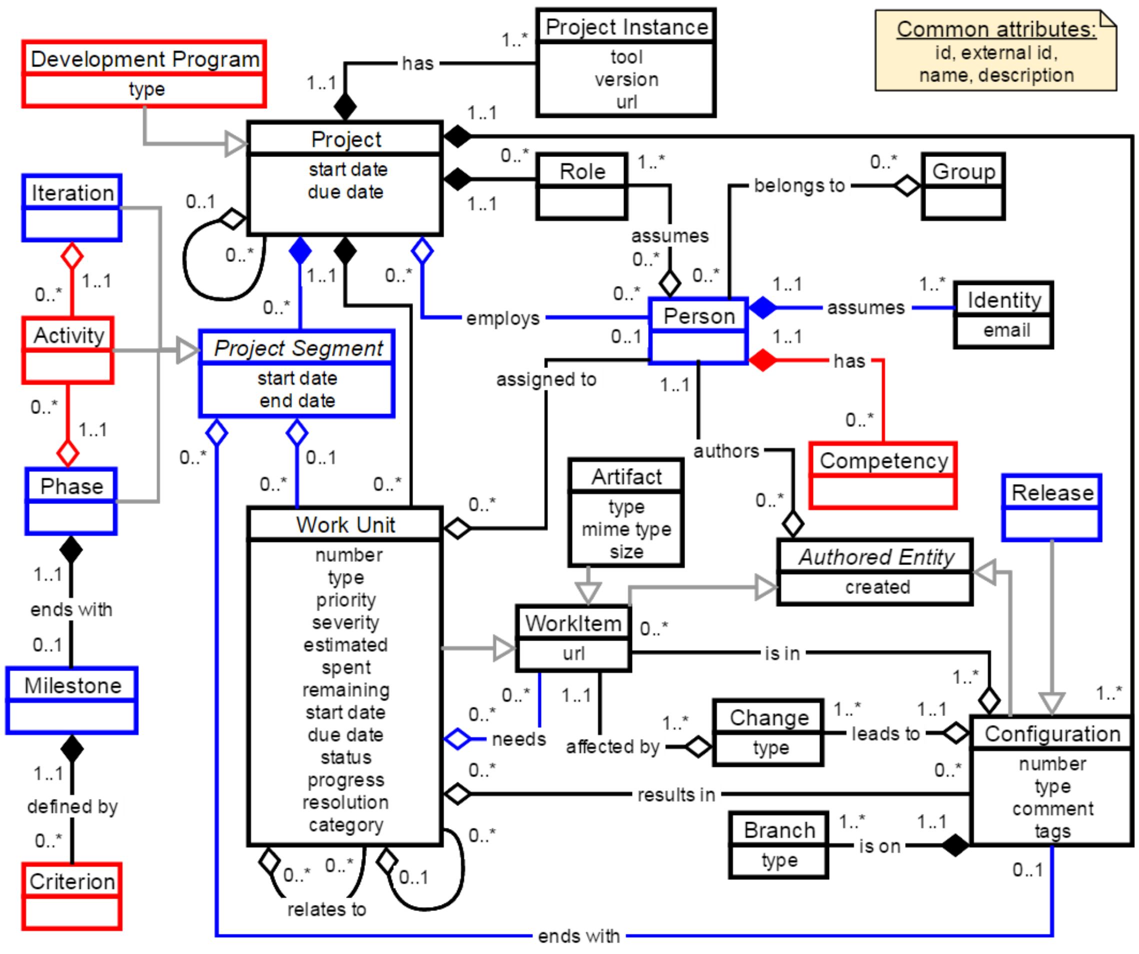

Fig. 1. SPADe domain metamodel.

and email addresses of project contributors (*identities*), their *roles* and associated *work units* (tasks). Blue entities need some analysis or user input (e.g., if there is only one mechanism in the tool to capture *project segments*, i.e. coherent parts of the project's timeline depending on a specific methodology used, we need to decide whether they are *iterations* or *phases*). The red entities present deeper specific challenges, for instance determining *activities* (sets of related tasks, e.g., performing product release) or personal *competencies*.

Even the analysis of the basic SPADe entities like *work units* (issues or tickets) and *configurations* (commits or revisions) can yield interesting and useful results. For example, only *work units* and *configurations* data are needed for checking over- and underestimation patterns or the practice of linking commits to issues, which many projects regard as good software engineering practice.

The metamodel is specifically constructed to allow for detecting defined best or bad practices (patterns and anti-patterns, smells) and deviations, and enables similar analysis with respect to social cooperation issues.

III. Codeface

The study of socio-technical patterns in software development through ALM data analyses can be approached in two ways. One is to focus on a specific tool and phenomenon first and widen the scope over time (bottom-up). Another starts with a general approach and implements it for specific tools (top-down). The former approach was adopted by the co-authors of this paper from the Laboratory for Digitalisation (LfD) at the Technical University of Applied Sciences (OTH) Regensburg in Germany.

We focus on the study of collaboration among developers in open source projects through social collaboration networks and their evolution over time; the tool supports various notions of "collaboration" like joint work in functions, commit review, and others. The goal of our research is to study the link between collaboration structures and product quality. We used Git VCS data and mailing lists as the primary sources of data. Weighted graphs describe, for instance, the relationship between core and peripheral developers and the turnover between these roles throughout the lifecycle of a project, as well as other aspects of open source development [1][9][10]. Fig. 2 shows illustrative examples for different collaboration structures with developers as nodes and identified collaborative relationships as edges.

The experimental tool supporting our analyses is called Codeface and is provided as Open Source Software (OSS). Its source code and documentation can be found on the supple-

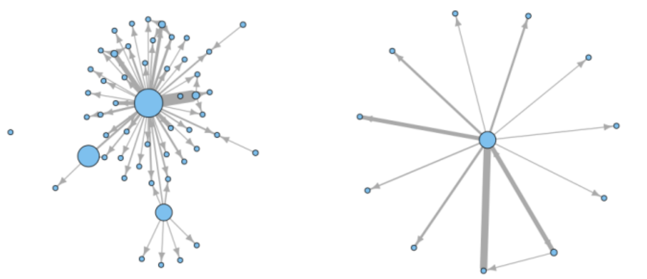

Fig. 2. Examples of colaboration network graphs obtained with Codeface. Nodes represent developers (size is proportional to their centrality) , and edges describe a collaboration relationship (e.g., joint work on a change).

mentary web page[4]. Though not yet specifically aimed at architects, once this role is identified in the collaboration graph, Codeface can provide comprehensive data on the measure of their collaboration with other team members and their temporal changes. However, current underlying data do not include the roles of the individuals.

The "Patch Stack Analysis" (PaStA) toolkit is another tool developed at LfD, used to understand socio-technical patterns. It also analyzes Git repositories and mailing lists, and focuses on the study of patch stacks (feature-granular modifications of mainline releases in parallel development). Namely it can identify patches moved from one stack to another, forward- and backports and invariants, and the time it took a particular patch to make it from a patch stack to mainline development [11]. Fig. 3 shows sample results of the analysis.

The next step in PaStA related research aims at the classification of patches by purpose (e.g. corrective, feature, etc.) to estimate their development cost and reuse potential, and to understand the implications of patches on software maintenance. The data, however, can be filtered to focus on the relationship between activity of a given team member and its effects on maintenance cost. Therefore, if the knowledge on the identity of an architect and the developers he/she collaborates with can be provided, the general process can be applied to data specifically tied to his/her activities in the project.

IV. SPADe

A complementary (top-down) approach to ALM tool data analyses is pursued by the paper co-authors at the Department of Computer Science and Engineering (DCSE) at the University of West Bohemia (UWB). It is based on the idea of performing process pattern analysis on ALM data converted and stored in a unifying metamodel [8], see also Fig. 1. The primary purpose is detecting well known and/or user defined bad practices (anti-patterns) in software development, and evaluating how a project follows defined process practices.

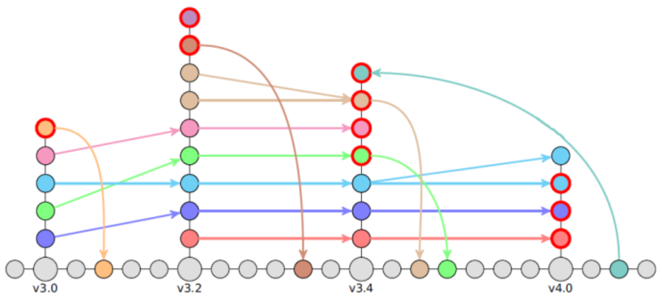

Fig. 3. PaStA identification of patch stacks (colored dots), back-/forwardports and invariant patches (indicated by the directed edges).

[4] https://github.com/siemens/codeface

From the perspective of studying architects' behavior in a given project, SPADe can serve multiple purposes. First it can analyze all available project data and assign the role of an architect to specific team member(s) based on that analysis, if the ALM tools used do not assign the role themselves. Second, it can store and aggregate data on architects' activities throughout the project from all the ALM tools used.

Finally, SPADe is designed to store and analyze issue (*work unit*) histories. Therefore, by analyzing changes in the *assignee* attribute of a *work unit*, comment content (stored as a *description* attribute of *change* entity) and authors, as well as issue relations, it can identify activities where an architect is involved, such as code reviews or architecture-induced refactoring request. This includes cases when the task is primarily assigned to someone else, or when the activity does not leave a mark in VCS or mailing list data.

V. Approaches and Tools Integration

Collaboration between the two research teams (OTH and UWB) entailed assessing the feasibility and mutual benefits of combining our research efforts. Due to the opposing strategies adopted by the two research tool sets (bottom-up for Codeface and PaStA, top-down for SPADe), neither of them was on its own able to fully grasp the complex project structures – including the specific issues surrounding the involvement of team roles such as architects. Therefore it proved desirable to enhance each of the tool sets with the current results of the other one.

On the database layer, a full integration was not yet desirable because of proprietary aspects of the respective databases that are not useful to the other party. The data models were however modified by drawing inspiration from one another. SPADe contributed to Codeface its common entities for data that are not tool-specific, in particular its section for issue-tracking data has been modified to use the *work unit* and *change* entities, relations between *change* and *field change* (an entity not visible in the domain model in Fig. 1), and between *work unit* and *configuration* (issue and commit). The attributes of a Codeface entity *issue*, previously heavily Bugzilla specific, were also changed to those of the SPADe *work unit* (a corresponding entity). This way, the Codeface model acquired the ability to capture the commit-issue relation.

Analysis of Codeface data model and algorithms clarified some of the more subtle details and challenges of mining Git repositories and mailing list archives. During these efforts it was found, that the respective data models were similar in important aspects, which at the same time simplified their modifications and informally validated the correctness of the SPADe metamodel.

Because both approaches use different technologies for their implementations (Codeface and PaStA are mainly based on Python and R, while SPADe utilizes Java) a straightforward full integration into a single toolset is currently not feasible, nor is it deemed necessary.

VI. Benefits

The enhancement of the Codeface and PaStA data models by SPADe common entities, which is already finished, allows us to capture ALM data from various tools and enhance our analyses with proper issue-tracking data for more complete and accurate results. The commit-issue relation will provide an opportunity for a more detailed and accurate determination of collaboration graphs and developer social networks.

One such case is the ability to study whether architectural activities have impact on product quality – for instance, the involvement of software architects in tasks and commits related to code refactoring or API development. Another research question enabled by the integration is to find out where architects are in reality positioned in the project collaboration structure – their prevailing communication with contributors in "developer" roles, can, for instance, signal a different methodology than closer links to contributors in project management roles.

Furthermore, Codeface will significantly benefit from the use of the data-mining ability of SPADe and its data pumps to collect data from multiple sources. SPADe data will also be used to include issue-tracking information into the PaStA patch classification, which will be useful in analyzing, for instance, maintenance activities. These can in turn be used to assess whether architect involvement has impact on the project.

The SPADe meta-model has been influenced in structurally minor, but nonetheless important ways by the Codeface data model mainly in the area of VCS and mailing list data. On the level of analysis, the approach to recognizing multiple identities (aliases, user names, email addresses, etc.) belonging to the same person has been adopted from Codeface to SPADe. This contribution will enhance the quality of SPADe-managed data, and will allow us to better identify potential architects. The PaStA classification of patches will provide a basis for determining change (work unit) classes in SPADe, which in turn will allow for better detection of architects' involvement in tasks of other members. Finally, SPADe will use the results provided by Codeface and PaStA as indicators for specific anti-patterns both in- and outside the scope of architects' activities.

VII. Conclusion

We have demonstrated the feasibility of defining a common data model for data from various ALM tools and its use for analyses of socio-technical patterns in software development projects, which is an important part of our research. The use of this common model in the integration of our teams' research tools, which we have practically validated and found mutually beneficial, will enhance their analytical capabilities. We have discussed that one of the possible enhanced uses is the evaluation of architects' behavior in software development and the impact of the role on different aspects of projects.

Our current focus is to concentrate on using open source projects as sources of experimental data, due to their straightforward availability, and to grow our dataset extensively both by mining new projects and accessing new sources (so far untapped ALM tools). It is our expectation that the resulting dataset based on multiple sources and a common metamodel can become a valuable resource for independent research on software projects and their socio-technical patterns.

Acknowledgment

The work was supported by the Ministry of Education, Youth and Sports of the Czech Republic – university specific research grant SGS-2016-018 Data and Software Engineering for Advanced Applications and project LO1506 (PUNTIS - Sustainability support of the NTIS Center).

References

[1] M. Joblin, S.Apel, and W. Mauerer, "Evolutionary trends of developer coordination: a network approach," Empirical Software Engineering, Springer, Nov. 2016, pp. 1-45, doi: 10.1007/s10664-016-9478-9.

[2] D. Draheim, and L. Pekacki, "Process-centric analytical processing of version control data," Principles of Software Evolution, Proc. of 6th Int'l Workshop on, 2003, pp. 131-136, doi: 10.1109/IWPSE.2003.1231220.

[3] D. M. German, "Mining CVS repositories, the softChange experience," Mining Software Repositories (MSR), Proc. of Int'l Workshop on, 2004, pp. 17-21.

[4] D. M. German, and A. Hindle, "Visualizing the evolution of software using softChange," Int'l Journal of Software Engineering and Knowledge Engineering, vol. 16(1), World Scientific Publ Co Pte Ltd., Feb. 2006, pp. 5-21, doi: 10.1142/S0218194006002665.

[5] G. Grambow, R. Oberhauser, and M. Reichert, "Towards a workflow language for software engineering," Parallel and Distributed Computing and Networks / Software Engineering (PDCN, SE), 10th Int'l Conf. on, ACTA Press, Feb. 2011, doi:10.2316/P.2011.720-020.

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

[7] P. Pícha, and P. Brada, "ALM Tool Data Usage in Software Process Metamodeling," Software Engineering and Advanced Applications (SEAA), 42th Euromicro Conf. on, IEEE, Aug.-Sep. 2016, pp. 1-8, doi: 10.1109/SEAA.2016.37.

[8] 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), Elsevier, Feb. 2014, pp. 103-116, doi:10.1016/j.infsof.2013.10.001.

[9] M. Joblin, W. Mauerer, S. Apel, J. Siegmind and D. Riehle, "From developer networks to verified communities: a fine-grained approach," Int'l Con. on Software Engineering (ICSE), Proc. of the 37th, IEEE Press, May 2015, pp 563-573, doi: 10.1109/ICSE.2015.73.

[10] M. Joblin, S. Apel, C. Hunsen and W. Mauerer, "Classifying Developers into Core and Peripheral: An Empirical Study on Count and Network Metrics," Int'l Conference on Software Engineering (ICSE), Proc. of the 39th, IEEE Press, May 2017.

[11] R. Ramsauer, D. Lohmann and W. Mauerer, "Observing Custom Software Modifications: A Quantitative Approach of Tracking the Evolution of Patch Stacks," Symposium on Open Collaboration (OpenSym), Proc. of 12th Int'l, ACM, Aug. 2016, doi: 10.1145/2957792.2957810.