A Project Monitoring Cockpit Based On Integrating Data Sources in Open Source Software Development

Stefan Biffl, Wikan Danar Sunindyo and Thomas Moser
Institute of Software Technology and Interactive Systems
Vienna University of Technology
Vienna, Austria
{biffl, wikan, tmoser}@ifs.tuwien.ac.at

Abstract — **Many open source software (OSS) development projects use tools and models that come from heterogeneous sources. A project manager, who wants to analyze indicators for the state of the project based on these data sources, faces the challenge of how to link semi-structured information on common concepts across heterogeneous data sources, e.g., source code versions, mailing list entries, and bug reports. Unfortunately, manual analysis is costly, error-prone, and often yields results late for decision making. In this paper we propose linking OSS data sources using semantic web technologies as foundation for providing integrated indicators project status analysis. We introduce the design concept of a project monitoring cockpit, ProMonCo, and evaluate the feasibility and effectiveness with a prototype for calculating communication metrics in a real-world context, the Apache Tomcat project. Major result was that ProMonCo efficiently supports frequent project monitoring by calculating communication metrics based on semantically integrated data originating from heterogeneous OSS project data sources.**

Keywords - open source software development; communication metrics; semantic integration

I. INTRODUCTION

Stakeholders of open source software (OSS) development projects routinely use a wide range of tools and models for developing and managing software products [1]. For example, developers may use SVN[1] for managing source code versions, mailing lists for the communication between developers, and bug reports for tracking software defects. In OSS development projects, with stakeholders who typically work in dispersed locations and time zones, project managers need to be able to manage and monitor the status of the project based on relevant indicators, e.g., by monitoring the communication level between the stakeholders. In healthy projects, the number of communication level between the stakeholders tends to be proportional, while the challenged projects show fluctuations which signify the imbalance of the number of communication level between the stakeholders [2].

Currently, project managers use quantitative measurement approaches for managing and monitoring the status of the projects [3], e.g., the number of commits to the SVN, the number of mailing list posts, or the number of issues in a bug report tool in a certain period. Further analyses to find related information between data sources like the correlation between the email conversation, bug status changes and SVN/CVS commits with the health status of the projects, need to be performed manually, which is costly, error-prone and often takes too much time for the analysis results to be useful for the project manager's decision making.

Communication metrics [4] can provide the foundation for relevant analyses and insights into software development processes, like finding the relationship between the project managers position and their communication forms with other stakeholders in the network. In comparison to more traditional software metrics based on software products, metrics generated from communication artifacts are fairly simple to compute and available early on in the software lifecycle begins. Insights from using communication metrics could not be achieved by traditional metrics, since they are based on different artifacts generated during a project that are not suited for indicating the quality of communication between team members. In addition, the free and widely available communication archives of OSS projects present a rich source for applying communication metrics.

Tools and models in OSS development projects typically come from heterogeneous sources. While the project stakeholders informally can agree on common concepts (e.g., project artifacts or projects issues) in the project, the representation of these common concepts usually varies across tools regarding the structure and terminologies (e.g., code and documentation artifacts, or bugs and project tasks) used. Therefore, expert know-how is needed to identify semantically equivalent concepts that look syntactically different. A project manager, who wants to analyze the state of the project based on information from these project data sources, faces the challenge of how to link semi-structured information (i.e., information that is available in a format both readable for humans and machines) on common concepts across heterogeneous data sources, e.g., source code versions, mailing list entries, and bug reports. While domain experts can conduct the linking manually, this manual analysis is typically costly and error-prone. Therefore, an integrated view of relevant project data is often not available to the project manager to support making relevant decisions, such as planning releases of new software product versions or restructuring the developer team.

[1] http://subversion.tigris.org

In this paper, we introduce a project monitoring cockpit, *ProMonCo*, which uses semantic integration approaches for bridging semantic gaps between heterogeneous project data sources as foundation for comprehensive automated analyses on the integrated project data model. We build on semantic web technology, the Engineering Knowledge Base (EKB) semantic integration framework [5], to explicitly link the data model elements of several heterogeneous OSS project data sources based on their data semantic definitions. The EKB consists of two types of ontology layers: the common domain knowledge layer and the local tool knowledge layer. To bridge the semantic gaps (different representations for equivalent semantic concepts) between these ontology layers, there are mappings between the local and domain ontologies that can query local knowledge using common domain knowledge syntax.

We propose a design concept for the project monitoring cockpit *ProMonCo* and implement a prototype to demonstrate how to calculate a set of communication metrics for the project manager. We evaluate the feasibility and effectiveness of the *ProMonCo* prototype by calculating three kinds of communication metrics based on real-world OSS development data originating from heterogeneous data sources in the Apache Tomcat[2] and other Apache projects. Major result is that *ProMoCo* efficiently supports frequent project monitoring by calculating communication metrics based on semantically integrated data originating from heterogeneous OSS project data sources.

This remainder of this paper is structured as follows: Section 2 summarizes related work on project monitoring and communication metrics, on the use of semantic web technologies for software engineering, and on OSS development. Section 3 identifies the research issues, while section 4 introduces the *ProMonCo* prototype, describes its architecture and the implemented communication metrics. Section 5 reports on the evaluation of *ProMonCo* in the context of the Apache Tomcat and other Apache projects based on the implemented communication metrics, and additionally discusses the findings. Finally, section 6 concludes the paper and identifies further work.

II. Related Work

This paper is based on previous work on project monitoring and reporting for OSS development projects. Manual reporting is a conventional approach that seems to fit well to tightly coupled organizations but is hard to apply for loosely coupled organizations like OSS development projects [6]. In this section, we summarize related work on project monitoring and communication metrics, on the use of semantic web technologies for software engineering, and on OSS development.

A. Open Source Software Development

OSS development is a knowledge-intensive domain that could be improved by using semantic web technologies. Sharma et al. [1] state that OSS development models are considered to be successful if they meet the original requirements regarding software quality. Even though some OSS projects do not finish successfully, in general they can be used as a model for proprietary software development as well. Sharma et al. [1] also introduced a framework for creating and maintaining hybrid-open source software communities, which consist of structures, processes and cultures. The wide range of stakeholders involved the different types of processes to be performed and cultural heterogeneities can lead to semantic gaps between the stakeholders, which need to be bridged for comprehensive tool support of OSS projects, e.g., by using semantic web technologies [7].

Mockus et al. performed an analysis on two OSS projects, namely the Apache Web Server and the Mozilla browser [8; 9]. Their goal was to quantify the aspects of developer participation, core team size, code ownership, productivity, defect density, and problem resolution for OSS projects. They collected and analyzed the data from different sources, for example developers' mailing list, concurrent version archive (CVS) and problem reporting database (BUGDB). However, they focused on the use of non-integrated historical artifacts originating from different data sources to measure the quality of software. In this paper, we integrate both historical and current artifacts originating from different heterogeneous project data sources to get more information (e.g., current project status, times needed for bug fixing, etc.) for decision making.

B. Project Monitoring and Communication Metrics

Von Krogh and von Hippel [3] investigate OSS development processes and found differences between monitoring commercial software development and OSS development. In commercial software development, the project manager can apply tight management of processes and take precautions to prevent employees from leaking software-related trade secrets and information to competitors, while in OSS development software architecture and functionality are governed by a community consisting of developers who can commit code to the authorized version of the software.

Yamauchi et al. [10] state that in a traditional perspective, managing and leading OSS development projects seems to be impossible, because no formal quality control program exists and no authoritative leaders monitor the development. For them it is surprising that the OSS development can achieve smooth coordination, consistency in design and continuous innovation while relying heavily on electronic environment as face-to-face supplementary, however, project monitoring for OSS projects is still in a very early phase. In addition, they discuss how OSS development avoids limitations of dispersed collaboration and addresses the sources of innovation in OSS development. Further research is needed to reveal how typical project management methodologies can be adapted to the OSS domain in order to improve the software quality, e.g., by monitoring typical OSS project product and process data.

Wahyudin et al. [6] discuss that project monitoring traditionally focused on human-based reporting, which is good for tightly coupled organizations to ensure the quality of project reporting. In loosely-coupled organizations like in OSS development projects, this approach does not work well because the stakeholders typically work voluntarily and flexibly. One way to measure the performance of the project is by correlating and

[2] http://tomcat.apache.org

analyzing process event data (e.g., mailing list artifacts or bug reports) from the OSS community.

Sharma et al. [1] observe the OSS development projects based on three aspects: namely structure, processes, and culture. In OSS processes, stakeholders can have governance mechanisms, for example by applying membership management, rules and institutions, monitoring and sanctions, and reputation as one of the prime motivators for the OSS developers. Even though membership in OSS projects is open to anyone, the OSS communities manage membership in conjunction with rules, institutions, monitoring, and sanctions. They illustrate how OSS projects can be monitored via social interaction and sanctions from the communities. However, the relationships between the project data produced by the stakeholders, the activities of the stakeholders, and the quality measurement of OSS were not analyzed in their study. We build on their research by performing social network analyses using heterogeneous data sources to monitor the project communication and project status of OSS projects.

Communication metrics [4] can provide the foundation for relevant analyses and insights into software development processes and in addition, the free and widely available communication archives of OSS projects present a rich source for applying such communication metrics. To our knowledge there is only very little research on communication metrics. Dutoit and Bruegge [4] found initial empirical evidence that metrics based on communication artifacts can generate significant insight into the status of an application development process in the context of commercial software development.. Communication metrics may even provide better insights (e.g., project status) in the development processes than code-based metrics, since they focus more on process than on product specific data. Current general practice is to measure and analyze the software code. However, this practice ignores that software code is only available late in the development process and not the only artifact generated. Communication artifacts, such as e-mails, mailing list entries, memo notes, or records generated by groupware tools are valuable information that are available early and can be used to investigate the health of the development process.

C. *Semantic Web Technologies for Software Engineering*

Semantic integration is defined as the solving of problems originating from the intent to share data across disparate and semantically heterogeneous data [11]. These problems include the matching of ontologies or schemas, the detection of duplicate entries, the reconciliation of inconsistencies, and the modeling of complex relations in different data sources [12]. One of the most important and most actively studied problems in semantic integration is establishing semantic correspondences (also called mappings) between vocabularies of different data sources [13].

The application of ontologies as semantic web technologies for managing knowledge in specific domains is inevitable. Noy and Guinness [14] note five reasons to develop an ontology, namely to share common understanding of the structure of information among people or software agents, to enable reuse of domain knowledge, to make domain assumptions explicit, to separate domain knowledge from the operational knowledge, and to analyze domain knowledge.

Happel and Seedorf [15] summarized the use of ontology-based approaches in different phases of the software engineering lifecycle. They compared research advances in software engineering and knowledge engineering, and found that the software engineering community has been focusing on software modeling, while the knowledge engineering community has been eager to promote several modeling approaches to realize the vision of the semantic web. Hence, both disciplines are closely related and the communities from both areas can contribute to each other. The paper categorized ontologies in software engineering into four areas based on time dimension (development time vs. run time) and types of knowledge (infrastructure vs. software), namely ontology-driven development, ontology-enabled development, ontology-based architectures, and ontology-ruled architecture. The authors saw the main advantage of ontologies in software engineering, among others, in the ability to define and use logic-based formalisms for software engineering in the context of the semantic web efforts. The flexibility of ontologies can make the combination of software engineering data from heterogeneous sources easier and more effective. While this paper is an important first step towards a better understanding of possible benefits of ontologies in software engineering, implementation issues remain open.

Moser et al. [5] introduced the Engineering Knowledge Base (EKB) framework as a semantic web technology approach for addressing challenges coming from data heterogeneity and applicable for different domains, in [5] for the production automation domain. Further, Biffl et al. [4] used the approach for solving similar problems in the context of OSS projects, in particular, frequent-release software projects. We build on this previous work by using and extending the proposed project data fetcher tool for extracting and integrating OSS project data from heterogeneous data repositories.

Other uses of ontologies in software engineering, e.g., for supporting software architecture decisions or for reuse of software development knowledge, have been suggested by Akerman and Tyree [1] and by Antunes et al. [2]. Since these decisions and reuse possibilities are also interesting for OSS development projects, the investigation of the applicability of ontologies for OSS development projects regarding these issues is also important.

III. RESEARCH ISSUES

In this paper, we focus on the aim of the project managers to improve the efficiency of project reporting and monitoring. The questions which can be raised are what the current situation of project reporting and monitoring in the OSS domain is, and how its efficiency can be improved. For answering these questions, the project manager needs to be able to collect communication data from development project and perform a set of communication metrics for further analysis purposes, like project reporting (e.g., average bug response time), project monitoring (e.g., monitoring the current project status or activities) or decision making (e.g., planning releases of new software product versions) on the OSS development. In compari-

son to more traditional software metrics based on software products, metrics generated from communication artifacts are fairly simple to compute and available early on in the software lifecycle begins. In addition, the free and widely available communication archives of OSS projects present a rich source for applying communication metrics.

Collecting related information from heterogeneous data sources for project reporting and monitoring is not an easy task for project managers, because each data source typically has its own proprietary formats and data structures. Semantic web technologies can help to solve problems originating from semantic heterogeneities among different data sources, by supporting the knowledge representation and storage for all project development data. The utilization of rule and reasoning can help the user to check the correctness of related data originating from heterogeneous data sources and find the relationships (e.g., synonyms, generality-specialty) between the data. From these challenges we derived the following research issues.

RI-1. Feasibility and effectiveness of tool support for project monitoring and reporting based on heterogeneous data sources in an OSS project context. On the foundation of general semantic web technology we investigate the design for tools that collect and semantically integrate data from OSS sources in order to build analysis applications suitable for project monitoring and reporting. In addition, we evaluate the feasibility to build a project monitoring cockpit *ProMonCo* as example for such an application. Finally, we evaluate the effectiveness of the *ProMonCo* tool to automate relevant aspects of project monitoring and reporting.

RI-2. Automated derivation of project health indicators from communication metrics from OSS project data sources. As application for the *ProMonCo* tool we investigate whether relevant project indicators, such as health indicators based on communication metrics like the *user coupling* metric and the *bug history* metric can be correctly and efficiently derived from heterogeneous data sources in OSS projects.

User Coupling Metric (UCM). The UCM is supposed to give significant insight into the communication structure of a project team [1]. Team members with significant importance for the will become very centrally located nodes. Project members that do not participate at the intra-team communication intensively will result in less central nodes. Very incommunicative behavior may even result in sub-graphs containing only a few nodes. After building the social network graph out of the available communication artifacts, several centrality measurements can be conducted. These measurements should indicate the importance of individual project members for the existing communication network.

Bug History Metric (BHM). The BHM will contrast the communication effort of the project members with the bug activities during a certain period of time. Further evaluation will show, if an increasing number of found or resolved bugs results in a growing amount of communication. If a correlation exists, project managers may consider supporting and intensifying the communication effort. This may e.g. be achieved by emphasizing team meetings.

To answer these research issues we gather requirements from interviews with OSS project experts and develop a prototype of the *ProMonCo* project monitoring cockpit. We then perform initial evaluations and data analyses using the data of the OSS Apache Tomcat project and other OSS Apache projects as a test-bed. We choose the Apache Tomcat project, because it has a long story of development, is quite stable, exists in several versions, and therefore provides wide variety of development data sources for analysis.

IV. The *ProMonCo* Project Monitoring Cockpit

This section describes the implemented Project Monitoring Cockpit prototype, called *ProMonCo*. First, the Project Data Fetcher Tool, used to collect and integrate OSS development project data, is described shortly. Then, *ProMonCo* is introduced and its architecture is explained. Additionally, the implemented communication metrics of *ProMonCo* are described.

A. Project Data Fetcher

Considering the requirement of semantically integrating data originating from heterogeneous OSS project data sources, a tool for the extraction of project data for Apache projects called Project Data Fetcher was initially developed [16]. This tool allows the gathering of project artifacts from the mailing list, the Bugzilla[3] database and the Subversion versioning system of Apache projects. The retrieved data is the basis of the evaluation of the designed communication metrics of this thesis. The Project Data Fetcher uses an ontology for the storage of the extracted project data. In contrast to a conventional database, an ontology is capable of a proper knowledge representation based on well-defined semantics. In addition, an ontology provides reasoning capabilities that may be utilized to discover previously hidden information by deducting new facts out of existing ones.

B. Project Monitoring Cockpit ProMonCo

The Project Monitoring Cockpit *ProMonCo* is an application that can be used for monitoring the status of project based on communication artifacts. By using *ProMonCo*, project manager can trigger the fetching of project data and calculate implemented communication metrics based on the fetched artifacts. As mentioned previously, *ProMonCo* uses the information gathered and integrated by the Project Data Fetcher. *ProMonCo* uses the generated ontologies for extracting the relevant information, generating communication metrics out of it and visualizing the outcome. For this purpose, the project manager has to load previously generated ontologies. All available communication metrics can be calculated automatically out of the data stored in the ontology, according to the settings selected by the user. Figure 1 gives an overview of the architecture of *ProMonCo*.

The architecture of *ProMonCo* consists of three layers, namely Input Handler, Communication Metrics Analyzer and Output Handler. In the Input Handler layer, the ontology input manager uses previously generated project ontologies as an input for *ProMonCo*. These ontologies will be processed and

[3] http://www.bugzilla.org

analyzed according to the different purposes and criteria of the communication metrics. The total communication metric uses the number of mail posted from the ontology in certain period and shows the visualization graphics depend on the time granularity and the chart style. The user coupling metric uses the interaction between developers by using communication means and shows the social network graph in the form of communication graph, betweenness centrality, closeness centrality and degree centrality. The bug history metric uses bug data from the ontology and shows bug activities diagram and capability to export file for further statistical analysis purposes.

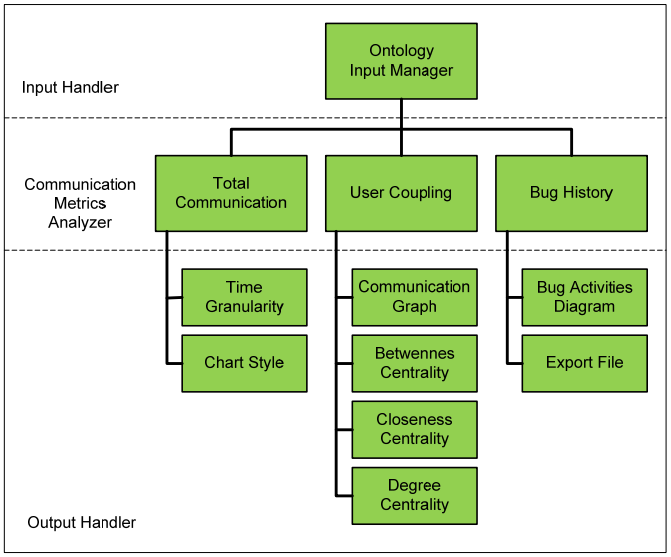

Figure 1. Overview *ProMonCo* architecture.

C. *Communication Metrics Implementation*

This section shortly summarizes the three types of implemented communication metrics, namely *Total Communication*, *User Coupling* and *Bug History*.

Total Communication Metric (TCM). The TCM is calculated by counting the number of communication artifacts during a certain period of time. The user can change the graphical representation and the calculated time period by changing the appropriate settings. The TCM is comparable to the traditional Lines of Code (LoC) metric, and serves as basis for more advanced metrics.

User Coupling Metric (UCM). The basic idea of the UCM is to create a social network graph out of the available mailing list information and hence acquire important insights in the communication behavior of the development team. The social network graph is supposed to reveal how deep each team member is involved in the overall communication. Each member of the development team is displayed by a vertex within the graph while the communication between members will be represented by an edge. Alongside the graph, several centrality measurements are calculated to determine numerically which members play a central role in the inner team communication. The first difficulty in the design of the UCM consists of the decision on how to build the edges displaying communication between team members. One possibility consists of building the edges between the vertexes when a mailing list user answers to a message of another user. However, the answers within a topic of the mailing list often do not only correspond to one message of another user, but rather to the topic in general. Therefore, all users participating in a topic are considered to be communicating with each other and will consequently be connected by edges in the communication graph.

Bug History Metric (BHM). The BHM is based on the TCM and aims at making the communication effort and bug activities comparable. The measurement compares the number of communication artifacts generated within a specified period of time with the amount of bug tracking entries. This comparison allows identifying correlations between these two activities and therefore provides vital information for project managers. If a high number of found or resolved defects results in a high communication effort, it is important to provide the project members with sufficient possibilities to communicate with each other, for example by planning more team meetings. The output of the BHM is a diagram that compares the communication effort with the corresponding bug activities for each time interval. To make these datasets visually comparable, the two plots are overlaid. This allows the user to observe the shape of the curves and to recognize if a correlation of some sort occurs. Additionally, further statistical tests can be conducted by using the export feature of *ProMonCo* to save all calculated results as CSV (comma-separated values) file.

V. Evaluation and Discussion

In this section, we discuss the evaluation of the project monitoring approach and the project monitoring cockpit *ProMonCo* regarding the research issues introduced in section 3.

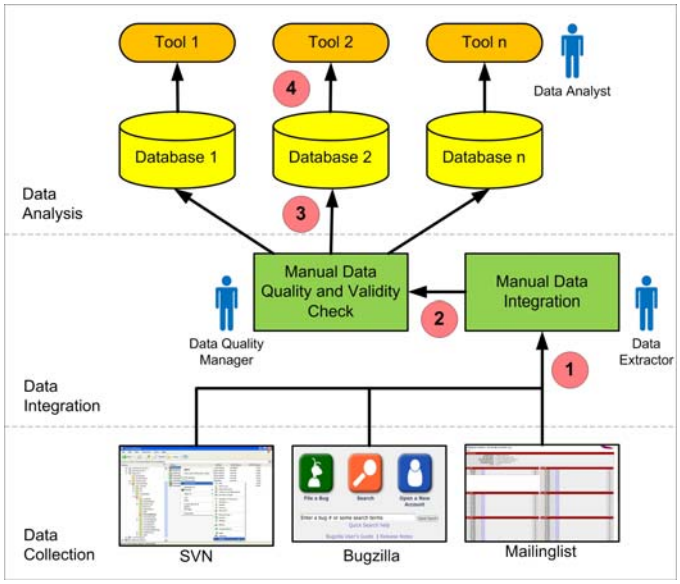

Figure 2. Manual Project Monitoring and Reporting.

Feasibility and effectiveness of tool support for project monitoring and reporting based on heterogeneous data sources in an OSS project context. Manual project monitoring and reporting has been used so far for commercial software development projects based on human reporting and structural organization [6]. However, this approach is not suitable for

loosely coupled forms of organizations like OSS projects that cannot rely on human reporting because of geographical distribution and primarily voluntary work of the project participants. Figure 2 shows the process of manual project monitoring and reporting and identifies limitations of this approach.

Heterogeneous data sources, such as SVN artifacts, Bugzilla data or mailing list entries, need to be manually integrated by a data extractor. The data extractor needs to identify the structure and the content of each data source, as well as their mutual relationships during the data integration process, but since the collected raw data may have incompatible formats, the data integration process can become inefficient and error prone (1). Current software quality monitoring approaches such as software defect prediction focus mainly on measuring software quality by means of static code metrics (2), such as code complexity, size and volume [17], which often fails to capture important distributed development process data (e.g., maturity level of code peer review prior to release), and in addition lacks of capability in providing early warning of certain quality status or risks due to the late data availability (i.e., too close to system deployment date). Furthermore, many studies treat all defects in similar way with no respect to defect severity level [18]. The results of this data integration need to be checked by a quality manager in order to guarantee the correctness, quality and validity of the integrated data before storing it in the database. There may exist several different database systems to support data analyses using different analysis tools (3). And finally, Project Monitoring typically is performed by reviewing the status and results of each analysis tool (4).

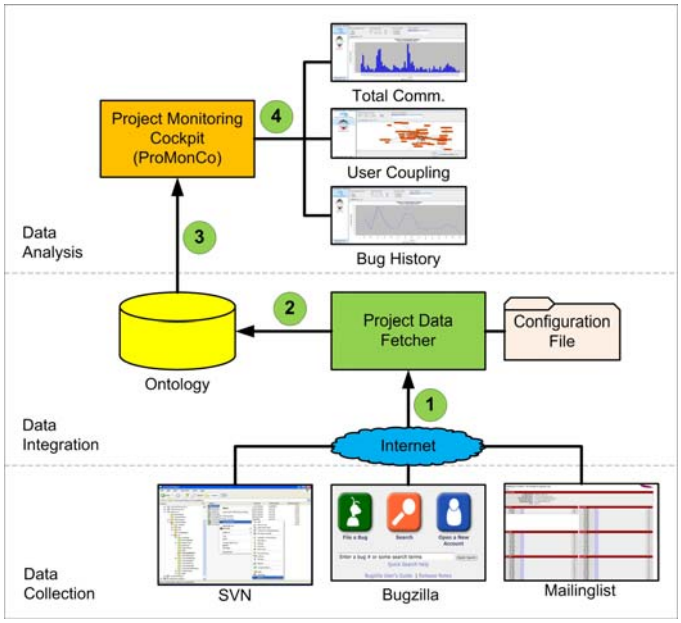

Figure 3. Project Monitoring and Reporting using *ProMonCo*.

Figure 3 shows the project monitoring and reporting process when using *ProMonCo* and identifies benefits of the approach. The Project Data Fetcher will fetches the data from the particular data sources via the web, semantically integrates the data based on a set of heuristics and then stores the integrated data in the project ontology [16], thus enabling a more efficient collection and integration of OSS project data (1) and putting together all required information with their relationships and rules into the project ontology for further analysis (2). The project ontology is then used as input for *ProMonCo* (3) to calculate several communication metrics used for monitoring and reporting of a project's status (4).

Automated derivation of project health indicators from communication metrics from OSS project data sources. *ProMonCo* provides three types of communication metrics to show the status of analyzed OSS projects: namely the *Total Communication* (TCM), the *User Coupling* (UCM), and the *Bug History Metric* (BHM).

The evaluation of the TCM shows that the amount of communication artifacts depends on the size of the project team. Projects with few participants produce evidently less mailing list entries than projects with a higher amount of developers. *ProMonCo* offers the possibility to compute the TCM for three different time intervals: yearly, monthly and weekly. The most meaningful time interval is the monthly view. On one hand, computing the metric for every week produces measurements with a very high standard deviation, making it hard to interpret the results. On the other hand, generating the metric on a yearly basis does not produce comprehensive information about the communication effort of the development team. Hence, calculating the metric for every month gives the project manager the best basis for observing the communication effort over time. Figure 4 illustrates a diagram of the TCM calculated monthly for six Apache projects, namely Tomcat, Ant, Cocoon, Lenya, Log4j and POI.

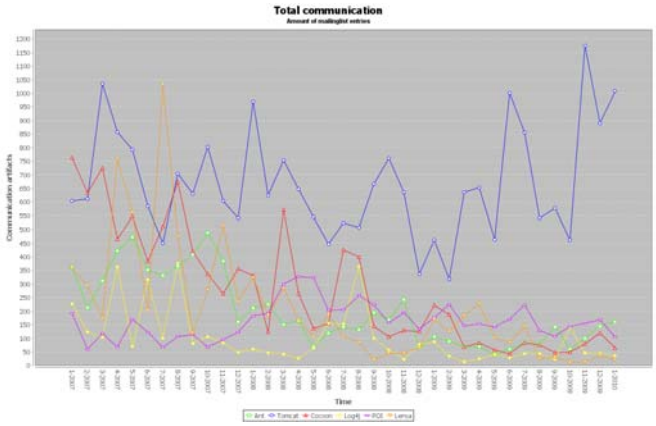

Figure 4. *ProMonCo* Total Communication Metric.

The UCM is based on principles and methods of social network analysis. It is used to generate a graph that visualizes the flow of communication between team members of a software project and to calculate the centrality measurements for each participant. The Apache Tomcat project was examined in regards of the communication behavior of the team members. For this purpose, the project data of the past three years was gathered using the Project Data Fetcher tool. The generation of the communication graph reveals that only very few sub graphs exist. These sub graphs only contain a very small amount of mailing list participants that are no core members of the project and do not commit source code to the project repository. According to the communication graph, all core project members are connected and communicating with each other. The graph

shown in Figure 5 represents the communication behavior of the Apache Tomcat project members that was calculated for the time period between November 2009 and February 2010.

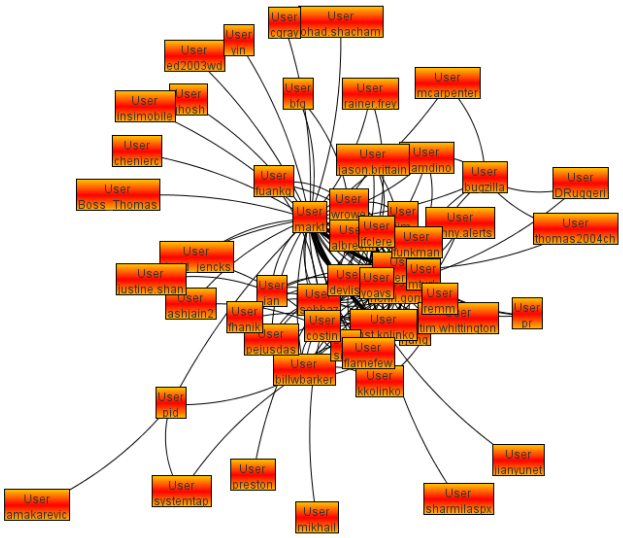

Figure 5. Communication Graph of the Apache Tomcat Project.

A closer look at the centrality measurements reveals that two persons have outstanding values compared to the rest of the development teams: Mark Thomas and Rainer Jung. Further research revealed that these two project members do not only hold a very central position within the communication network, but are also involved in the project management. Although both members lie very central, the point betweenness centrality value for Mark Thomas is extraordinarily high. An actor within a social network that has a high betweenness centrality value is supposed to have a significant influence on the flow of information. Individuals in such central positions have the possibility to coordinate the processes of the group and should therefore be part of the project management [19]. Hence, the high betweenness centrality value for Mark Thomas indicates that he possesses a high potential in influencing and coordinating the rest of the development team.

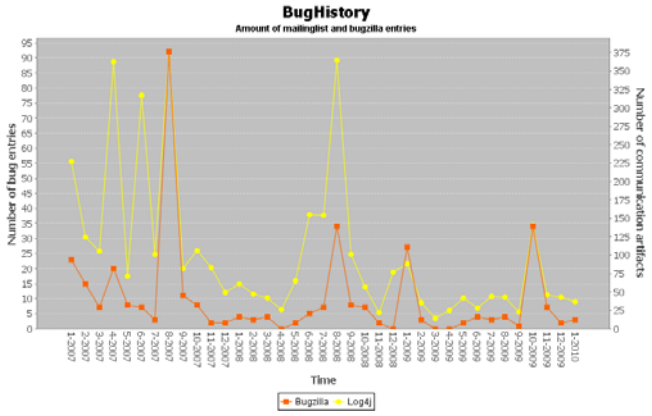

Figure 6. Resolved bugs and mailing list entries of the Apache Log4j Project.

The BHM compares the amount of bug tracking activities to the communication effort within a certain period of time. It reveals a possible correlation between Bugzilla artifacts and the amount of mailing list entries generated by the development team. *ProMonCo* calculates the metric for new found bugs and for resolved bugs separately. Hence, project managers are able to see if the search and discovery of bugs or the correction of defects results in an increased amount of communication. An obvious correlation between Bugzilla and mailing list artifacts may be recognized visually. Figure 6 shows a line chart of the resolved bugs and the amount of mailing list entries between January 2007 and January 2010 for the Apache Log4j[4] project. For the evaluation of the BHM, the results of the Apache projects were investigated with SPSS.

For testing the possible correlation between bug tracking artifacts and mailing list entries, the two-tailed Pearson correlation is used. The evaluation of the results shows a significant correlation between Bugzilla entries and the number mailing list messages in nearly all projects. The correlation occurs regardless of comparing the number of new found or resolved defects with the amount of communication artifacts. The only exception is the Apache Ant project which does not show a significant correlation between resolved bugs and mailing list usage.

Table 1 contains the results of the correlation testing for new found bugs, whereas Table 2 shows the results for resolved bugs.

Table 1. Pearson correlation for new found bugs.

	n	Pearson Correlation	Significance (two-tailed)
Ant	37	0.704*	0.000
Lenya	37	0.885*	0.000
Log4j	37	0.696*	0.000
POI	37	0.706*	0.000
Tomcat	37	0.689*	0.000

*Correlation is significant at the 0.01 level (two-tailed).

Table 2. Pearson correlation for resolved bugs.

	n	Pearson Correlation	Significance (two-tailed)
Ant	37	0.018	0.916
Lenya	37	0.808*	0.000
Log4j	37	0.706*	0.000
POI	37	0.746*	0.000
Tomcat	37	0.438*	0.007

*Correlation is significant at the 0.01 level (two-tailed).

The significant correlation between the amount of Bugzilla and mailing list artifacts may be explained by the excessive use of the mailing list for the discussion of topics regarding defects. In addition, all comments a team member creates in Bugzilla concerning a bug are automatically forwarded to the mailing list. A high number of new or resolved bugs usually result in a

[4] http://logging.apache.org/log4j

high number of Bugzilla comments that are sent to the mailing list. Therefore, considering different sources of communication artifacts may result in a different outcome of the BHM.

VI. Discussion and Conclusion

Many OSS development projects use tools and models that come from heterogeneous sources. A project manager, who wants to analyze indicators for the state of the project based on these data sources, faces the challenge of how to link semi-structured information on common concepts across heterogeneous data sources, e.g., source code versions, mailing list entries, and bug reports. Unfortunately, manual analysis is costly, error-prone, and often yields results late for decision making.

In this paper we proposed linking OSS data sources using semantic web technologies as foundation for providing integrated indicators project status analysis. We introduced the design concept of a project monitoring cockpit, *ProMonCo*, and evaluated the feasibility and effectiveness of *ProMonCo* with a prototype for calculating communication metrics in a real-world context, the Apache Tomcat project as well as other Apache projects.

Major result was that *ProMonCo* efficiently supports frequent project monitoring by calculating communication metrics based on semantically integrated data originating from heterogeneous OSS project data sources. The usage of the Project Data Fetcher to collect and semantically integrate the data originating from the different heterogeneous OSS project data sources allows for more efficient project monitoring and reporting. ProMonCo builds up on this integrated project ontology and can be used to efficiently and effectively calculate a set of communication metrics which can be useful for monitoring the project status and for reporting to project or quality managers. Communication metrics like the User Coupling Metric (UCM) or the Bug History Metric (BHM) can reveal new insights on OSS projects. In the evaluation, we used the UCM to identify project members of the OSS Apache Tomcat project which are also involved in the project management, based on centrality measurements of the social network communication graph resulting from the UCM. Furthermore, using the BHM, we identified a significant correlation between Bugzilla entries and the number mailing list messages in nearly all evaluated Apache projects. The correlation occurs regardless of comparing the number of new found or resolved defects with the amount of communication artifacts.

Future Work will include the integration of additional project data sources, such as different sources of communication artifacts, in order to be able to perform new types of communication metrics, as well as also other process metrics. Furthermore, we plan to expand our approach to other OSS project types, such as SourceForge[5] projects.

Acknowledgment

The authors want to thank Raphael Zaki and Stefan Huber for implementing prototypes of the Project Data Fetcher and of *ProMonCo*. This work has been supported by the Christian Doppler Forschungsgesellschaft and the BMWFJ, Austria. In addition, this work has been partially funded by the Vienna University of Technology, in the Complex Systems Design & Engineering Lab.

[5] http://sourceforge.net

References

[1] S. Sharma, V. Sugumaran and B. Rajagopalan, "A framework for creating hybrid-open source software communities," Information Systems Journal. City, vol. 12, pp. 7-25, 2002.

[2] D. Wahyudin, K. Mustofa, A. Schatten, S. Biffl and A. M. Tjoa, "Monitoring the "health" status of open source web-engineering projects," International Journal of Web Information Systems. City, vol. 3, pp. 116-139, 2007.

[3] G. von Krogh and E. von Hippel, "Special issue on open source software development," Research Policy. City, vol. 32, pp. 1149-1157, 2003.

[4] A. H. Dutoit and B. Bruegge, "Communication metrics for software development," IEEE Transactions on Software Engineering. City, vol. 24, pp. 615-628, 1998.

[5] T. Moser, S. Biffl, W. D. Sunindyo and D. Winkler, "Integrating Production Automation Expert Knowledge Across Engineering Stakeholder Domains," Proc. 4th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2010), IEEE Computer Society, 2010, pp. 352-359.

[6] D. Wahyudin and A. M. Tjoa, "Event-Based Monitoring of Open Source Software Projects," Proc. Proceedings of the The Second International Conference on Availability, Reliability and Security, IEEE Computer Society, 2007, pp. 1108-1115.

[7] S. Biffl, W. D. Sunindyo and T. Moser, "Bridging Semantic Gaps Between Stakeholders in the Production Automation Domain with Ontology Areas," Proc. 21st International Conference on Software Engineering and Knowledge Engineering, 2009, pp. 233-239.

[8] A. Mockus, R. T. Fielding and J. Herbsleb, "A case study of open source software development: the Apache server," Proc. 22nd Intl Conference on Software engineering, ACM, 2000, pp. 263-272.

[9] A. Mockus, R. T. Fielding and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," ACM Trans. Softw. Eng. Methodol. City, vol. 11, pp. 309-346, 2002.

[10] Y. Yamauchi, M. Yokozawa, T. Shinohara and T. Ishida, "Collaboration with Lean Media: how open-source software succeeds," Proc. ACM conference on Computer supported cooperative work, ACM, 2000, pp. 329-338.

[11] A. Halevy, "Why your data won't mix," Queue. City, vol. 3, pp. 50-58, 2005.

[12] N. F. Noy, A. H. Doan and A. Y. Halevy, "Semantic Integration," AI Magazine. City, vol. 26, pp. 7-10, 2005.

[13] A. Doan, N. F. Noy and A. Y. Halevy, "Introduction to the special issue on semantic integration," SIGMOD Rec. City, vol. 33, pp. 11-13, 2004.

[14] N. F. Noy and D. McGuinness, 2001, Ontology Development 101: A Guide to Creating Your First Ontology, Stanford Knowledge Systems Laboratory.

[15] H.-J. Happel and S. Seedorf, "Applications of Ontologies in Software Engineering," Proc. 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006) held at the 5th International Semantic Web Conference, 2006, pp. 1-14.

[16] S. Biffl, W. D. Sunindyo and T. Moser, "Semantic Integration of Heterogeneous Data Sources for Monitoring Frequent-Release Software Projects," Proc. 4th International Conference on Complex, Intelligent and Software Intensive Systems, IEEE, 2010, pp. 360-367.

[17] N. Fenton and M. Neil, "A Critique of Software Defect Prediction Models," IEEE Trans. Softw. Eng. City, vol. 25, pp. 675-689, 1999.

[18] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus and P. Gruenbacher, Value-Based Software Engineering: Springer Verlag, 2005.

[19] L. C. Freeman, "Centrality in social networks conceptual clarification," Social networks. City, vol. 1, pp. 215-239, 1979.