Observability of Software Engineering Processes in Open Source Software Projects Domain

Wikan Danar Sunindyo
Christian Doppler Laboratory for Software Engineering Integration for Flexible Automation Systems
Vienna University of Technology
Favoritenstrasse 9-11/188 1040 Vienna, Austria
+43 588 01 - 18801
wikan@ifs.tuwien.ac.at

ABSTRACT
Open Source Software (OSS) projects as a complex software engineering system is an ideal domain for empirical software engineering, because it provides a lot of data and possibility to introduce new approaches that can be easily adapted and enrich methods to improve the software quality. As flexible and continuous developing systems, OSS projects are always growing as new requirements from users and new code from developers come into the project. However, at some point the project manager wants to know the status of OSS project whether the project is in the right direction and the product is delivered in a good quality. To get the immediate status of OSS project, some efforts have been done to observe the software engineering processes of OSS project. However, current approaches focus on limited areas of health indicators of OSS project. In this research, we will improve the approaches by proposing a framework that integrates different approaches on observing the software engineering processes and monitoring the health status of OSS project. Our objectives are to define observability factors of software engineering by making literature research on prior works and to assess the OSS engineering processes using the observability factors to show that this approach is working and can improve the software quality. In this research we use OSS projects domain as our context, and other engineering domains as references, e.g., production automation and (software+) engineering domains. Our contributions are improvement on the data collection and the data analysis steps.

Categories and Subject Descriptors
software, software engineering, process metrics, product metrics.

General Terms
Management, Measurement.

Keywords
Software Engineering Process, Observability, Open Source Software Project.

1. INTRODUCTION
In this section, we discuss motivation, rationale, background, and advice needed for this research.

1.1 Motivation
In the complex software engineering systems domain, the software engineering processes hold an important role to determine the quality of the systems in general. A lot of research works have been done to improve the software engineering processes to improve the quality of the systems as well. However, their diffusion in practice has been disappointing [1], they are still lack of observability of improvement that could be seen to strengthen the argument for improvement application in the engineering processes.

The observability of software engineering processes can make the processes are easier to be seen and evaluate by the observers and other system stakeholders. So, the observers can set the goal of observation of the processes, collect the processes data, and then analyze the data for process improvement.

In this research, we use Open Source Software (OSS) development project domain as a use case for empirical study on observability of complex systems. We use some approaches, like health indicators [2], communication metrics [3], and social network analysis [4] to observe the status of OSS project.

1.2 Rationales
Some rationales to have a research on observability of the complex software engineering systems are as follows: to guarantee the quality of the software in each step, to observe the status of the software project, and to obtain the status of software project during development phase for further decision making, e.g., whether certain project is sustainable and is worth supporting in the case of OSS project.

1.3 Background
The Open Source Software (OSS) development approach as an alternative to the conventional (proprietary) software development gets attention from the people of software engineering field as a promising approach to improve the software quality [3].

The OSS development environment provides bunch of data that can freely and easily to access by the quality manager to inspect and monitor the status of the project, by applying certain analysis approach to the data, e.g., by using communication metric or defect prediction approaches. New approaches also can be applied and adapted to the OSS projects, make the methods to improve the software quality in the software engineering field becomes more rich than before. This is also useful to improve the empirical

software engineering area and is quite promising to be applied in conventional software development as well.

The role of OSS development in the empirical software engineering area for example, possibility to improve the methods to collect data, analyze data, and present data in fashionable way, that could make people understand better on how the software quality can be improved by using OSS development approach rather than by using the conventional software development approach.

Even though there are a lot of researches concerning the OSS development project for improving the software quality in empirical software engineering area [5, 6], measuring the software quality, e.g., determining the status of OSS project by using process data is still challenging research are to explore.

1.4 Use Case

The OSS projects can be seen as a flexible and continuous system, since they will always grow bigger and bigger as the development of new requirements and feedbacks from users and new source codes and features from the developers. The OSS can always be developed and improved [7]. Some stable versions can be released by the project manager after fixing most of bugs, but some more features can always be added in the next version. So, no final state is clear enough from OSS project, to get information of the project only from its product data.

However, at some points the project manager wants to assess the status OSS project to know whether the project is on the right track and direction and whether the product is delivered in the good quality, so the users like to use it and satisfy with the quality of the OSS product. The project manager also concerns on whether the project is sustainable and is worth to support [2].

From literature research, we found that the healthy project consists of core developers, codevelopers, active and passive users in onion shape that have different roles to guarantee that the project is worth to be done and the product is useful [8].

While the project is continuously developed, the final and stable product is rarely happened, which means the software version is just released after certain period. Measuring the quality of OSS only based on the final and stable product will take a long time, the quality manager should wait until all processes are finished and product released. While the requirement for inspecting and monitoring the status of OSS project is inevitable. To obtain the status of OSS project faster, we can use the processes data derived from OSS project, instead of just use data from final product.

1.5 Advices Needed

Some advices that could support this research are for example, how to collect data for this research, how to analyze the collected data by using advanced methods, and how to present the data analysis results properly.

2. RELATED WORK

This section provides some related works on observability, software process improvement some approaches on observing the OSS project status.

2.1 Observability

From Merriam Webster dictionary[1], *observability* is defined as *capable of being observed*. *Observed* itself is defined as *to watch carefully especially with attention to details or behavior for the purpose of arriving at a judgment*.

In early diffusion of innovations work, Rogers [9] identifies five perceptions about an innovation that affects the rate of diffusion of the innovation: relative advantage, compatibility, complexity, observability, and trialability.

In [9], observability is defined as *the degree to which the results of an innovation are visible to others*. The easier it is for individuals to see the results of an innovation, the more likely they are to adopt it. Such visibility stimulates peer discussion of a new idea, as friends and neighbors of an adopter often request innovation-evaluation information about it.

2.2 Software Process Improvement

The search for new ideas and innovations to improve software development productivity and system quality continues to be a key focus of industrial and academic research. Fichman and Kemerer [10, 11] define software process innovations simply as any new way of constructing application software products: technologies that change an IT group's process for developing software applications.

Some approaches on improving the software process have been done by using empirical research methods. One of them is from Dyba [12], whose proposing an instrument for measuring the key factors of success in software process improvement. He defines instrument construction as the process of developing the data collection device in order to define and obtain relevant data for a given research question. By defining the instrument for measuring the key factors of success in software process improvement, we can use it also in OSS project context. Some key factors of SPI success are namely, business orientation, leadership involvement, employee participation, concern for measurement, exploitation of existing knowledge (learning by experience), and exploration of new knowledge (learning by experimentation). Because the OSS development processes has differences with conventional software development processes, we should put more concern on the details of the differences.

Green [1] has examined factors that motivate developers to adopt and sustain use of SPI. She defines a SPI's "visible" success in terms of (1) perceptions of how easy it is to use, (2) perceptions of improved productivity, (3) perceptions of improved quality, (4) perceptions of the usefulness of the SPI. Through a combination of quantitative and qualitative analyses, she has verified critical success factors for the success of a SPI to software developers. However, this research is focusing only one SPI (i.e., Personal Software Process) evaluation, such that generalizability across SPIs is limited.

2.3 OSS Status Observing Approaches

The health indicators approach is introduced and improved by Wahyudin et al [2, 13]. This approach is focusing on the using of developer contribution proportion and bug service delay to measure the health status of OSS project.

[1] http://www.merriam-webster.com/

The communication metrics approach to measure the status of OSS project is proposed by Biffl et al. [3] A project monitoring tool has been built based on the communication metrics on OSS project data and produced three different metrics, namely total communication metric, user coupling metric, and bug history metric.

The total communication metric focuses on the illustration of all collected communication data (e.g., mailing list entries) from the OSS project developers in certain period and displayed in certain intervals (e.g., weekly, monthly, or yearly). This metric is useful to understand the relationship between the amounts of communication entries within certain period of time following certain time interval. The result shows that the size of developers influence the number of communication entries. Bigger size of developers means more communication entries for some periods.

The user coupling metric is based on the relationships between the stakeholders of OSS project. It can illustrate the centrality of the users related with their role in the project. More central position of the users means more important their role in the project, e.g., become the project manager or the core developer.

The bug history metric 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.

The application of social network analysis (SNA) [14] to determine the status of OSS project is also proposed by Kaltenecker [4]. In his research [4], SNA can show some centralities of OSS developers that related with the risk of the project.

3. RESEARCH OBJECTIVES, QUESTIONS AND HYPOTHESES

In this section, we discuss the objectives of the research, some questions that we will answer by proposing some hypotheses to test. We follow the GQM (Goal, Question Metric) method [15, 16] to do this research.

3.1 Research Objectives

The objectives of our research in observability of software engineering processes in complex systems are as follows.

RO-1: Definition of observability factors of software engineering. We define the observability factors of software engineering processes. What kind of factors from software engineering processes that can be derived to determine the quality of the systems.

RO-2: Assessing the observability factors of the software engineering processes. By using those factors defined in the first objective, we assess the observability factors of the software engineering processes to show that this approach can improve the quality of the systems.

To investigate the observability of software engineering processes, we will conduct empirical studies by using OSS development projects as a case study.

The OSS development projects consist of a lot of project data that could be useful for analysis on software quality improvement. From our recent work, we found several observability factors that can be used to determine the status of OSS project, for example the health indicators approach that used the proportion of developer contribution and bug service delay [2].

We also use our experience for observing software engineering processes in (software+) engineering systems domain [17] and production automation domain [18].

In this case we will improve the observability factors of OSS project processes, e.g., by repairing the health indicators of OSS project, by adding new parameters, data sources, metrics and tool for monitoring the indicators.

3.2 Research Questions

Some research questions can be derived from the research objectives are as follows.

RO-1: Definition of observability factors of software engineering. From our prior works and related researches, there are a lot of findings shows that observability on engineering process can support us to suggest for possible improvements. However, the observability approaches we developed so far are still on limited domains. We want to answer questions; is there any common observability factors that could be suitable almost for all engineering processes? if not, what kind of observability factors that could be suitable for one engineering processes and not for others, what circumstances that could make the observability factors suitable to one engineering processes. For answering this question, we will use OSS project domain as a use case.

RO-2: Assessing the observability factors of the software engineering processes. After getting the suitable observability factors for our case, then we conduct our experiment on assessing the software engineering processes for improving our system quality.

Some questions can be raised here. Can the observability factors improve the quality of the system? If we observe the software engineering process, collect the process data and then make data analysis on it, is the quality of the system increasing? How the observability factors that can be used to shorten project cycle time, decrease costs and decrease risks? What kind of decisions that could be useful to improve the quality of the system?

3.3 Research Hypotheses

Based on the previous research objectives, we can arrange some hypotheses to answer the research questions as follows.

RO-1: Definition of observability factors of software process engineering.

Since we have been doing some works and literature research on observability factors in software engineering processes for different domains, we find out that the observability factors is useful for certain domain. We want to generalize this method to other engineering systems. To do this, we will examine which observability factors are useful and could improve the system quality. We will apply these factors in OSS development project domain. For the component of quality, we use van Solingen's

process improvement focus on quality, namely risk, cost, and time [15] and define the hypotheses as follows.

$H1_0$: a good observability factor can decrease the risks of OSS project development, e.g., by providing defect prediction for the project.

$H1_1$: a good observability factor can not decrease the risks of OSS project development, e.g., by providing defect prediction for the project.

RO-2: Assessing the observability factors of the software engineering processes.

From our works and literature research on observability factors in OSS project domain [2-4], we find out that the different approaches sometimes covering overlap areas. We want to discover how the relationships between different observation approaches and whether the results from different approaches can strengthen our conclusion on OSS project status. We formulate these questions into these hypotheses.

$H2_0$: the classifications of health status results from different observation approaches are always the same and strengthen the conclusion of OSS health status.

$H2_1$: the classifications of health status results from different observation approaches are not always the same, so we can not derive the conclusion of OSS health status only by analyzing results from two different approaches.

4. RESEARCH APPROACH, EMPIRICAL STUDY DESIGN AND ARRANGEMENTS

In this section, we discuss research approach, design and arrangement for empirical study, some metrics that are used for this research, data analysis methods and techniques and validity threats and their control.

4.1 Research Approach

The research on observing OSS project is done by following these steps.

Step 1: Literature study on OSS project development. We make a literature study on OSS project development and different methods on how to observe the OSS project development, for example by using the process data analysis and organizational analysis.

Step 2: Definition of data to collect. We specify variables of data to collect. Based on these variables, we collect and validate data from project data sources to ensure sufficient quality to construct a framework for process observation.

Step 3: Appropriate data analysis approach. In this step, we commit appropriate analysis approaches based on the collected and validated data from step 1. For example, by using health indicators approach we make analysis on two indicators, namely the service delay and the proportion of the developer contribution. In OSS project, we define the service delay as a function of time to respond and time to resolve an issue/bug. The proportion of the developer contribution is very important to obtain an outlook of software creation performance and the current developers' motivation state. Both indicators are derived from the aggregation of metrics, which at the lowest level the metrics are obtained by mining the project repositories.

Step 4: Results in the empirical study. The last step in our framework is to evaluate the results by using empirical study. We can discuss the results with OSS experts for comparing different results from different approaches.

Figure 1 shows the research approach environment and research steps minus the literature study. The observation on processes is done by collecting process data from different data sources (SVN, mailing list, bug report), integrating data, analyzing data, and giving suggestion to improve OSS product and processes.

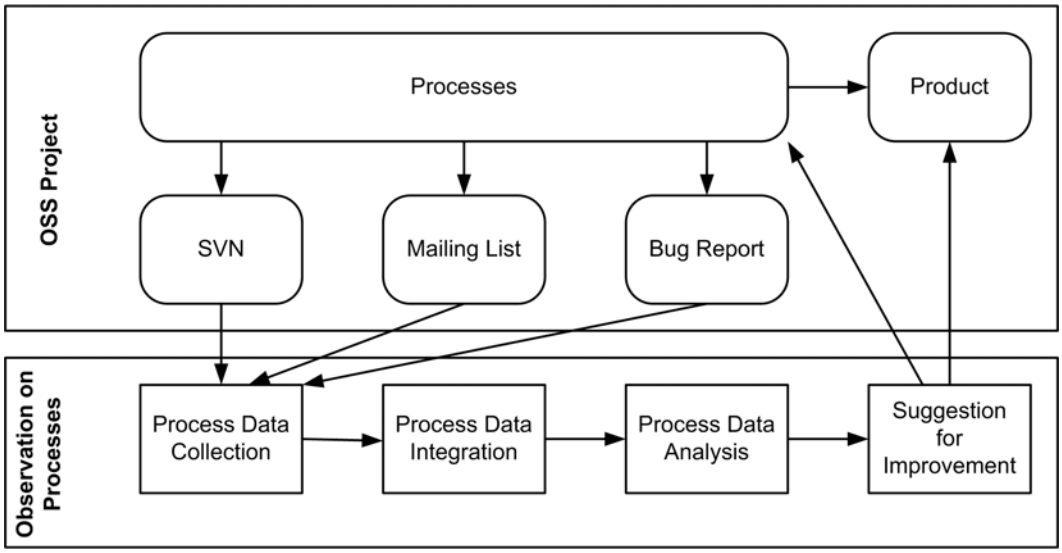

Figure 1. Research Approach Environment

4.2 Empirical Study Design and Arrangements

In this section, we describe a framework for observing software engineering processes of OSS projects, our empirical study objects, and data collection procedure from OSS projects repositories.

4.2.1 A Framework for Processes Observation

In this section, we describe a method framework for software engineering processes of OSS projects observation, which consists of four layers, namely (1) project tool and data source, (2) core framework, (3) framework plug-in/module analysis, and (4) presentation layer. This framework integrates all observation approaches and supports our empirical study. Figure 2 illustrates the framework for observing OSS processes.

Project Tools and Data Sources. In this layer we provide different kinds of repositories of OSS projects as data sources for the next layers. An OSS repository is a place where large amounts of source code of OSS are kept. They are often used by multi-developer projects to handle various versions and developers submitting various patches of code in organized fashion.

We provide different kinds of repositories of OSS projects as data sources for the next layers. An OSS repository is a place where large amounts of source code of OSS are kept. They are often used by multi-developer projects to handle various versions and developers submitting various patches of code in organized fashion.

Core Framework. In this layer, we provide several methods for data preparation before analysis. The preparation phase is including data collection, data cleaning, data integration and data validation.

We can provide different tools to collect and integrate data. By determining common characteristics of data in this layer, we can support the processes analysis in the next layer.

Framework Plug-In/Model Analysis. Different processes observation analysis approaches are provided in this layer. By using these approaches we can analyze the processes data collected in previous layer. Currently the analysis approaches are developed separately and have their own focus of observation. By proposing this framework, we expect to have integration of different analysis approaches that can be useful to strengthen the conclusion on the status of OSS project.

Some current analysis approaches are including health indicators [2], communication metrics [3], and social network analysis [4] approaches. We can add other analysis approaches simply by putting the new approaches into our framework and adapting the input from the data collection and data integration processes.

Presentation Layer. This layer consists of several approaches and tools to present the analyses results that can be useful for the project manager, the quality manager, or other stakeholders and systems. The presentation could be in the form of websites or report, or tools like project monitoring cockpit, or interfaces to connect to other application like Decision Support Systems or Executive Information Systems.

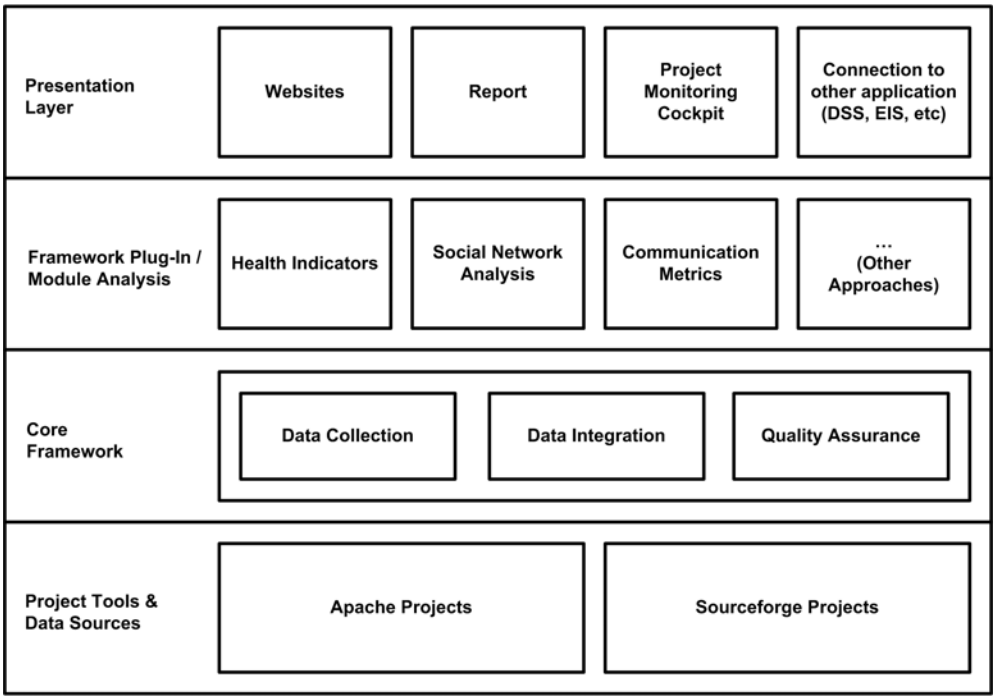

Figure 2. A Framework for observing OSS processes

4.2.2 Study Project Selection

For conducting the research, we select several projects from different project repositories, like Apache and Sourceforge. We also use data from active and non-active projects to compare the results and show the significant differences of the results from active and non-active projects if we use different kind of analysis approaches. The project management of different project repositories could also affect the results, for example the Apache projects are stricter, while the Sourceforge projects are more flexible. The descriptions of each project used in this research are as follows.

The health indicators approach is applied to four cases of large-matured Apache projects indicated by more than 20 contributors (core and peripheral developers) per project which three times outsized the average number of developers in most of OSS projects [19] and has already, at least, one major release (1.x, 2.x, etc). The set consists of two well known Apache projects (Tomcat V.5 and HTTP Server/HTPPD V.2), and two challenged projects (Xindice and Slide). We focused our evaluation process on the two health indicators: the proportion of developers' participation and the bug service delay. The origins of the four projects are as follows.

Apache Tomcat is a servlet container that is used in the official reference implementation for the Java Servlet and Java Server Pages technologies whose code base and specifications are donated by Sun under Java Community Process in 1999.

Apache HTTP Server is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT.

Apache Xindice is a database tool designed from the ground-up for storing XML data or what is more commonly referred to as a native XML database.

Apache Slide is a part of the Apache Jakarta projects. Slide is a content repository which can serve as a basis for a content management system/framework and other purposes.

The total communication metrics (TCM) approach is applied to six Apache projects, namely Tomcat, Ant, Cocoon, Lenya, Log4j and POI. 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.

The social network analysis approach used four Sourceforge projects. Gutenprint provides a package of free printer drivers for various POSIX-compliant operating systems like Linux or Mac OS. Due to it's initial focus the highest usage can be found in open source systems like Linux. Gutenprint supports about 700 different printers.

SQuirreL SQL Client is a graphical SQL client developed in Java. It is possible to view the structure of a JDVC compliant database, browse the data in tables ad use SQL commands to perform queries.

HSQLDB is a relational database engine developed in Java. Advantages are described as being a fast and multithreaded engine and server with features like transaction isolation and encryption.

XDoclet enabled attribute-oriented programming in Java but with the introduction of annotations in Java version 5.0 this project became obsolete.

Table 1 shows the summary of different projects used in project monitoring researches.

4.2.3 Data Collection Procedures

In collecting OSS processes data, we use two approaches, namely "manual" approach and automatic approach. A "manual" approach is when we collect data directly from the project website or by using some dedicated tools, like SVN, developer mailing list website, or bug report website. The example of the using of "manual" approach in collecting data is by Wahyudin et al. [2], which collect data from project website and development tools. Actually, the "manual" approach here is not really a manual approach, since it also uses tools to collect and integrate data. The term "manual" is used to differentiate with automatic approach and because the way to make relationship between interrelated data from different sources in "manual" approach is done manually.

Contrast to automatic approach, we build and use, like Project Data Fetcher, to collect and integrate the project data [3, 20]. The idea of the Project Data Fetcher tool is using ontology to automatically collect the project data from different sources (SVN, mailing list, bug report) and automatically integrate and match the related data, for example the same name in different data sources refers to the same person. This tool is useful to find further relationships between different data sources that are not easily discover by using "manual" approach.

Some relevant data that are needed to be collected from different tools are as follows. For SVN tools, we need to collect the name of project, version, added lines, deleted lines, log message, and revision. For mailing list, we collect subject, message, attachment and size of the message. For bug report tool, we collect the id of bug, status, type and priority.

4.3 Definition of Metrics

For our research on observing the OSS project status, we define some metrics for software quality measurement. Li et al. [21] classified metrics to construct software quality improvement models into four basic categories:

- *Product metrics* measure attributes of intermediate and final software products, e.g, lines of code (LOC) and number of classes. Product metrics are the most common predictors and supported by [22-24] as important predictors in closed-source development
- *Process metrics* measure attributes of development processes, e.g., project events (new bugs reported into bug tracker), state changes (bug status changed from unresolved to resolved), and activities such as, the number of files changed, and LOC churned per developer within a release. Mockus et al. [6], Weyuker et al. [25], and Nagappan et al. [26] suggested these metrics as important predictors. Some studies also called this group of metrics as *development metrics* [25], *project metrics* [7], or *changes metrics* [27].

- *Deployment and usage metrics* measure attributes of the deployment context and usage patterns of software releases, e.g., time since first release and time to next release [21, 28].
- *Configuration metrics* measure attributes of software and hardware configuration that interact with the software product/release during operation, e.g., the operating system supported by the release and the type of software application [24].

Table 1. Different projects used in processes observability researches

Project Name	Host	Stable release	Date release	Status	Written in	Type	Website
Health Indicators							
Tomcat	Apache	7.0.0	29-Jun-10	Active	Java	Servlet container HTTP web server	http://tomcat.apache.org/
HTTP Server	Apache	2.2.15	6-Mar-10	Active	C	Web server	http://httpd.apache.org/
Xindice	Apache	1.1	8-Dec-07	Retired	Java	XML Database	http://xml.apache.org/xindice/
Slide	Apache	5.0 M4	19-Dec-07	Retired	Java	Content management system	http://jakarta.apache.org/slide
Communication Metrics							
Tomcat	Apache	7.0.0	29-Jun-10	Active	Java	Servlet container HTTP web server	http://tomcat.apache.org/
Ant	Apache	1.8.1	7-May-10	Active	Java	Build Tool	http://ant.apache.org/
Cocoon	Apache	2.2.0	15-May-08	Retired	Java	Web application framework	http://cocoon.apache.org/
Lenya	Apache	2.0.3	20-Jan-10	Active	Java/XML	Content management system	http://lenya.apache.org/
Log4j	Apache	1.2.15	29-Sep-07	Retired	Java	Logging Tool	http://logging.apache.org/log4j
POI	Apache	3.6	14-Dec-09	Active	Java	API to access Microsoft Office formats	http://poi.apache.org/
Social Network Analysis							
Gutenprint	Sourceforge	5.2.5	11-Feb-10	Active	Java	Free software printer drivers	http://gimp-print.sourceforge.net/
SQuirreL	Sourceforge	3.1	3-Mar-10	Active	Java	Database administration tool	http://www.squirrelsql.org/
HSQLDB	Sourceforge	2.0.0	7-Jun-10	Active	Java	RDBMS	http://hsqldb.org/
Xdoclet	Sourceforge	1.2.3	5-May-05	Retired	Java	Attribute-Oriented Programming for Java	http://xdoclet.sourceforge.net/

4.4 Data Analysis Methods and Techniques

For analyzing data, we will use some methods like correlation analysis and Bayesian Network analysis.

The first step before statistical evaluation is the preparation of submitted data, including check for consistency. We use descriptive statistics to investigate the variation of dependent variables and independent variables.

- *Bar-Charts*: We use bar-charts to visualize the variation of basic results, e.g., the number of communication entries regarding the period of time.
- *Box-Plots*: To show variations of dependent variables, we apply *box-plots*.
- *Scatter Plots* display pair-wise samples in two dimensions to identify potential correlations.

To determine significant differences of dependent variables, we apply statistical tests at a significance level of 95%. In most cases we use the parametric t-test or the non-parametric Mann-Whitney Test to compare to samples within the experiment results to verify/falsify our research hypothesis. We also use other analysis methods like correlation analysis and Bayesian Network analysis.

4.5 Validity Threats and Their Control

For internal validity, the problem like missing data or incomplete data for analysis will be solved by using cleaning data approach before data analysis.

For external validity, the problems are including different project repositories for data collection and different tools used in collecting data. To solve these problems we find out the characteristics of similar data, rather than individual project data and propose a framework for integrating data.

5. SUMMARY

Currently we are working on the integration framework for processes observation that can support the integration of different approaches on analyzing process data of OSS project. Different approaches that could be used for observing the processes data are health indicators [2], communication metrics [3], social network analysis [4], process mining [17], and organizational mining [18]. By using these approaches, we can also make analysis on data to be collected.

The next step is including collecting the data and making empirical study for OSS project observation by using different project repositories and different analysis approaches.

6. REFERENCES

[1] Green, G. C., Hevner, A. R. and Webb Collins, R. The impacts of quality and productivity perceptions on the use of software process improvement innovations. Information and Software Technology, 47, 8 2005, 543-553.

[2] Wahyudin, D., Mustofa, K., Schatten, A., Biffl, S. and Tjoa, A. M. Monitoring the "health" status of open source web-engineering projects. International Journal of Web Information Systems, 3, 1/2 2007 2007, 116-139.

[3] Biffl, S., Sunindyo, W. D. and Moser, T. A Project Monitoring Cockpit Based On Integrating Data Sources in Open Source Software Development. In Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering (to publish) (Hotel Sofitel, Redwood City, San Francisco Bay, USA, July 1 - July 3, 2010, 2010).

[4] Kaltenecker, A. Deriving Project Health Indicators of Open Source Software Projects using Social Network Analysis. Vienna University of Technology, Vienna, 2010.

[5] Wahyudin, D., Schatten, A., Winkler, D. and Biffl, S. Aspects of Software Quality Assurance in Open Source Software Projects: Two Case Studies from Apache Project. In Proceedings of the Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (2007). IEEE Computer Society.

[6] Mockus, A., Fielding, R. T. and Herbsleb, J. D. Two case studies of open source software development: Apache and Mozilla. ACM Trans. Softw. Eng. Methodol., 11, 3 2002, 309-346.

[7] Wahyudin, D., Schatten, A., Winkler, D., Tjoa, A. M. and Biffl, S. Defect Prediction using Combined Product and Project Metrics - A Case Study from the Open Source "Apache" MyFaces Project Family. In Proceedings of the 34th Euromicro Conference Software Engineering and Advanced Applications (Euromicro SEAA 2008) (2008). IEEE Computer Society.

[8] Crowston, K. and Howison, J. The social structure of free and open source software development. First Monday, 102004.

[9] Rogers, E. Diffusion of Innovations. Free Press, New York, 1983.

[10] Fichman, R. G. and Kemerer, C. F. Adoption of software engineering process innovations: the case of object orientation. Sloan Management Review, 34, 2 1993, 7-22.

[11] Fichman, R. G. and Kemerer, C. F. The illusory diffusion of innovation: an examination of assimilation gaps. Information Systems Research, 10, 3 1999, 255-275.

[12] Dyba, T. An Instrument for Measuring the Key Factors of Success in Software Process Improvement. Empirical Software Engineering, 5, 4 2000, 357-390.

[13] Wahyudin, D., Schatten, A., Mustofa, K., Biffl, S. and Tjoa, A. M. Introducing "Health" Perspective in Open Source Web-Engineering Software Projects, Based on Project Data Analysis. In Proceedings of the IIWAS International Conference on Information Integration, Web-Applications and Services (Yogyakarta Indonesia, 04.12.2006 - 06.12.2006, 2006). Austrian Computer Society.

[14] Wasserman, S. and Faust, K. Social Network Analysis : Methods and Applications. Cambridge University Press, 1994.

[15] Van Solingen, R. and Berghout, E. The Goal/Question/Metric Method. McGraw-Hill Education, 1999.

[16] Basili, V., Caldiera, G. and Rombach, D. H. The goal question metric approach. 1994.

[17] Sunindyo, W. D., Moser, T., Winkler, D. and Biffl, S. Foundations for Event-Based Process Analysis in Heterogeneous Software Engineering Environments. In Proceedings of the 36th Euromicro Conference on Software Engineering Advanced Applications (SEAA 2010) (Lille, France, 2010).

[18] Sunindyo, W. D., Moser, T., Winkler, D. and Biffl, S. Process Analysis and Organizational Mining in Production Automation Systems Engineering. Christian Doppler Laboratory for Software Engineering Integration for Flexible Automation Systems, Vienna University of Technology, Austria, Vienna, Austria, 2010, http://www.ifs.tuwien.ac.at/files/u290/Process_Analysis_and_Organizational-techrep.pdf.

[19] Manenti, F., Comino, S. and Parisi, M. From Planning to Mature: on the determinants of open source take off. Universita Degli Studi Di Trento, Trento, 2005.

[20] Biffl, S., Sunindyo, W. D. and Moser, T. Semantic Integration of Heterogeneous Data Sources for Monitoring Frequent-Release Software Projects. In Proceedings of the 4th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2010) (2010). IEEE.

[21] Li, P. L., Herbsleb, J., Shaw, M. and Robinson, B. Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc. In Proceedings of the 28th International Conference on Software engineering (Shanghai, China, 2006). ACM.

[22] Basili, V. R. and Perricone, B. T. Software Errors and Complexity: An Empirical Investigation. Communications of the ACM, 271984.

[23] Denaro, G. and Pezze, M. An Empirical Evaluation of Fault-Proneness Models. In Proceedings of the International Conference on Software Engineering (ICSE2002) (2002). IEEE.

[24] Mockus, A., Weiss, D. and Zhang, P. Understanding and Predicting Effort in Software Projects. In Proceedings of the International Conference on Software Engineering (ICSE 2003) (2003).

[25] Weyuker, E. J., Ostrand, T. J. and Bell, R. M. Using Developer Information as a Factor for Fault Prediction. In Proceedings of the Third International Workshop on Predictor Models in Software Engineering (2007). IEEE.

[26] Nagappan, N., Ball, T. and Zeller, A. Mining metrics to predict component failures. In Proceedings of the 28th International Conference on Software Engineering (Shanghai, China, 2006). ACM.

[27] Moser, R., Pedrycz, W. and Succi, G. A Comparative Analysis of the Efficiency of Change Metrics and Static Code Attributes for Defect Prediction. In Proceedings of the 30th International Conference on Software Engineering (2008). ACM.

[28] Li, P. L., Shaw, M., Herbsleb, J., Ray, B. and Santhanam, P. Empirical evaluation of defect projection models for widely-deployed production software systems. SIGSOFT Softw. Eng. Notes, 29, 6 2004