MONITORING THE"HEALTH" STATUS OF OPEN SOURCE WEB ENGINEERING PROJECTS

Dindin Wahyudin[1,2], Khabib Mustofa[1,3], Alexander Schatten[1], Stefan Biffl[1], A Min Tjoa[1]

[1]Institute of Software Engineering and Interactive Systems,
Vienna University of Technology, Vienna, Austria
[2] Bandung Institute of Technology, Bandung, Indonesia
[3] Gadjah Mada University, Yogyakarta, Indonesia

Received: March 2007; Revised: June 26 2007

Abstract—*In response to the increasing number of open source software (OSS) project initiatives and the increasing demand of OSS products as alternative solutions by industries, it is important for particular stakeholders such as the project host/supporter (e.g., Apache Foundation, Sourceforge), project leading teams, and prospective customers to determine whether a (new) project initiative is likely to sustain and worthwhile to support. From a software project management point of view, a typical web-based OSS project can be viewed as a web engineering process, since most OSS projects exploit the benefits of a web platform and enable the project community to collaborate using web-based project tools and repositories such as mailing lists, bug trackers, and versioning systems (CVS/SVN) to deliver web systems and applications. These repositories can provide rich collections of process data, and artifacts which can be analyzed to better understand the project status. This paper proposes a concept of "health" indicators and an evaluation process that can help to get a status overview of OSS projects in a timely fashion and predict project survivability based on the project data available on web repositories. For initial empirical evaluation of the concept, we apply the indicators to well known web-based OSS projects (Apache Tomcat and Apache HTTP Server) and compare the results with challenged projects (Apache Xindice and Apache Slide). We discuss the results with OSS experts to investigate the external validity of the indicators.*

Index Terms—Open source project health indicator, Open Source Project Management, Web engineering, Web Mining, Information extraction, Communities on the web.

I. INTRODUCTION

Open source software (OSS) has caught our attention by the success and quality of its projects on the market, despite the fact, that its development does not follow traditional software development principles. In certain software product classes OSS offers comparable or better contributions than "closed source" commercial software products, making OSS a considerable alternative in many domains reaching from operating systems over web-frameworks and databases to office applications. However, many OSS projects are still in their initial phase, in an immature state or have reached the end of their life cycle, which means their survival seems heavily uncertain.

Our study begins with the question on how to improve the chances that an OSS project can stay "healthy" and to predict the chances for survivability for the next time periods (months or one quarter years ahead) or to reach success. Nowadays a great number of

OSS projects exists but for the individual it is hard to judge, whether it is worthwhile to take the effort of a closer look as a project.

In OSS, the project survival is a result of many underlying (connected) processes and can not be easily determined. Just to give an example: developers are typically not paid for their work, but contribute voluntarily on their own motivation basis. Hence, the dynamic of the development process is much more difficult to estimate compared to that of a typical commercial project. This issue is problematic for certain stakeholders in OSS community to fulfill their goals such as prospective customers of OSS products in order to decide which products will provide long-term warranty and enhancements; the hosting project (e.g. Apache and Eclipse) to provide or to continue support for some projects under their umbrella; project leading teams who steer the project's direction based on project status in timely fashion. For this reason, the Apache Software Foundation (ASF) has to decide whether or not to accept new project initiatives and set them up the *Apache Incubator*.

The scale of the problem is escalating when a large number of projects should be monitored in parallel. The resulting OSS project monitoring faces ever-increasing demands to provide pertinent data from the dynamics of the projects, to help stakeholders cope with complex masses of data/information, to provide competitive discriminators based on the stakeholder values; and to provide the "health" status of the project.

We address these needs by proposing a concept and evaluation of "health indicators" in open source projects. The basic argument for the strategy of our approach is derived from the analysis of literature and published studies. We apply the indicators to well-known OSS web-engineering projects for initial empirical evaluation of the concept. We perform project data analysis on the data retrieved from several successful OSS projects and the challenged ones. We discuss the results with OSS experts to investigate the external validity of the indicators.

II. RELATED WORK

This section briefly introduces recent practices and related works that support the implementation of health monitoring in OSS web-engineering projects.

A. *Open Source Software Project as a Form of Web Engineering*

Web engineering represents a new discipline in software development, as its major concern is to develop web using the emerging internet technology. Based on the definition of web engineering by (Ginige, Murugesan and Hansen, 2001), an open source web engineering project (such as Apache Tomcat, Apache HTTP Server, Phyton, and Mozilla) can be seen as an extreme form of web engineering practices, as it heavily relies on exploiting the web platform and web based tools during its development process. Current trend shows an increasing number of OSS products adoptions in the web engineering community. Just to give an example, Netcraft[1] Web Server Survey discloses that more than 70% of the web sites on the Internet are using Apache HTTP Server.

[1] http://news.netcraft.com/archives/web server survey.html (accessed at 20/02/2007)

Furthermore, our observations on 178,951 projects listed in Sourceforge[2] reveals that the top 5 project categories are Internet application (15.4%), Software development (15.1%), System (12.4%), Communication (10%), and Game/Entertainment (9.3%). Sourceforge ranks these projects into several categories: Planning (18,156 projects), Pre-Alpha (15,314 projects), Alpha (17,190 projects), Beta (23,198 projects), Production/Stable (19,531 projects), Mature (1,675 projects), and Inactive (2,124 projects). Although there is no formal definition from Sourceforge for these categories, these facts depict that most of the projects are web-related applications which are still in their early stages or already at the end of their lifecycle, and only a small portion (less than 2%) of the projects have reached maturity means they already have several stable releases.

Capiluppi et al., (Capiluppi, Lago and Morisio, 2003) suggest that typical OSS development seizes several problems such as the lack of requirement elicitation, no ad-hoc development process, less attention to quality and documentation, and poor practices of project management. Another typical characteristic is the high level of distribution, as OSS is built by a potentially large number of volunteers who are geographically distributed. It is worth noting, however, that currently a number of important OSS projects are supported by companies and some participants are not volunteers (e.g., JBoss, Apache JackRabbit, or OpenOffice). Although the development process in OSS project seems less organized, a recent study (Valverde, Theraulaz, Gautrais, Fourcassi and Sol, March-April 2006) suggested that the social structure in OSS projects could provide some hierarchy or meritocracy (such as in Apache) of management and controlling based on self-organizing patterns. This makes OSS projects interesting objects for the empirical study of web/software project management.

B. Software Project Monitoring

To obtain the actual status of the project, software project monitoring is required to supply timely, accurate and comprehensive project information as the basis for analysis and decision making. During the development, the monitoring process should balance the observation from both (a) time relevant process events data and (b) product-relevant artifacts data as the complement to each other. This combination will provide more accurate and less biased project information. In general, there are two monitoring approaches: *tool-based* and *human-based* monitoring. The tool-based approach seems most suitable for monitoring frequently a large number of process events data, or when human resources for monitoring are hard to obtain. On the other hand, the human-based approach seems favorable for a weekly/monthly process such as personal reporting as the summary of activities and progress status, and go to more detail if necessary by directly interviewing the team member.

Traditional software project management (Phillips, 2004; Royce, 2000) focuses on tracking formal achievements such as the progress and financial obligation and analyzing the merit of project participants based on routine personal reports and deliverables. As the consequence, most of the traditional project management is human-based monitoring, which often misses the process and information during its project execution. This can be

[2] http://sourceforge.net/ (accessed at 18/02/2007)

very risky; if a problem occurs, as Keil et.al (Keil, Smith, S.Pawlowski and Jin, 2004) found, participants tend not to report the actual condition of the project. Hence, additional data for comprehensive balanced reporting are needed before and during a crisis for raising issues well in advance to identify and to mitigate project risks.

Current trends in distributed software development such as open source web engineering project and web information system have led to new challenges regarding the scalability and expressiveness of methods (Schewe and Thalheim, 2005), especially when the project has to face (1) a large amount of process data to be monitored, (2) shortage of human resources for monitoring, and (3) most importantly, the loosely coupled project community as the result of a global project work. Consequently, monitoring such a system using only a human based approach is likely to be costly, time consuming, and error prone.

In this situation, a project leader should rely not only on human-based reports and project artifacts, but also supplement these sources of information with tool-supported process event data monitoring. Tool-driven health monitoring system has been successfully adopted by industries, reaching from hybrid system to business activity process monitoring (Thai, Pekilis, Lau and Seviora, 2001), which in principle can be adopted in software project management domain. Other study (Schatten, Tjoa, Andjomshoa, and Shafazand, 2001) suggested a web-based OSS tools for project monitoring, particularly for dislocated or distributed development such as OSS projects.

Open source software projects consist of large sets of process data residing in some projects' repositories. To extract useful information from such data repositories we need web data mining, starting with data preparation which may involve data cleaning, data transformations, selecting subsets of records and in case of data sets with large numbers of variables - performing some preliminary feature selection operations to bring the number of variables to some manageable range.

Several studies have used process data of the existing open source projects to better understand the aspects of successful distributed development. These studies observed the OSS projects by manually or by tool supported mining project repositories such as mailing lists, bug database (Mockus, Fielding and Herbsleb, 2002), SVN/CVS (German, 2004), and changes log (Chen, Schach, Yu, Offutt and Heller, 2004). The results clearly portrayed the development process pattern and the importance of community involvement in OSS projects. However further works are required to better understand the OSS project, to distinguish different status of projects and to estimate the project survivability.

III. THE CONCEPT OF OSS HEALTH MONITORING

Web engineering, in particular in an OSS project, is a process with a long life cycle. To ensure a project's survival, a decision maker needs to evaluate health status and recognize early symptoms of illness. Such indication could be obtained by correlating measures that are available during the development. Our research focused on shaping the "health" concept for OSS web-engineering project, later we proposed several health indicators, based on measurement of project performance metrics reside in OSS projects repositories.

A. *The "Body" of Open Source Project*

Crowston et al (Crowston, Annabi and Howison, 2003) suggested the success factors of an OSS project consisting of *software creation/developer contribution, software use intensity* and *software quality*. However to better illustrate the impact of these factors and typical risks during the development process, in our context; we model the OSS project as a body consisting of three major components: *the developer community, the user community* and *the software product*. We picture in Figure 1 the success factors and risks as interrelated states and activities which indicate the project component status. We assume that the survivability of the project is the result of the state of well being (liveness) of both communities indicated by facilitating rapid creation and deployment of the incremental product releases or patches[3]. Furthermore, this release should satisfy relevant user needs. The following subsection describes an analysis about our model in more detail, based on literature review and web observation.

Fig. 1. Causal Model of an OSS project. The status of an OSS project is the result of interrelated factors residing within the project communities and the product quality.

3 http://www.catb.org/ esr/halloween/ (accessed at 20/02/2007)

1) Developer Community Liveness

Successful OSS projects are not one time event. It is a process of a long life cycle which was first coined by Eric Raymond as “scratching the developer itch”. The developers’ community continues to contribute, develop, enhance, maintain and release the products **developer contribution**, iteratively in a typical OSS project management style. Therefore, in order to survive a project should attract more developers and boost their **motivation**.

Studies on the projects hosted in Sourceforge by (Krishnamurthy, 2002;Manenti, Comino and Parisi, 2005) signify that the developer community may consist of one developer only (as a single fighter) up to more than 250 developers at one time. They also disclose that 86.2 % of the projects employ less than 6 developers during the development processes.

A survey from The Boston Consulting Group (2002) disclosed that the developers’ motivations to join an OSS project are to stimulate their intellectual, to enhance their skills or to have access to the source code and user needs. As most of the developers join the OSS project for their self satisfaction, the degradation of developer motivation are not trivial and very likely to cause the project into deep problems if there is no appropriate counter measure. Lerner (Lerner and Triole, 2002) suggested that developers’ loyalty can be obtained by giving some incentives such as opportunity to contribute, community attention and recognition based on merit to the project.

According to Gacek (Gacek and Arief, 2004), developer in OSS project is comprised of *code/peripheral developers* and *core developers* which also called as *committers*. A project can generally be well developed and provide regular releases, which are appreciated by the user community, but it might actually be driven by very few active committers (**dependencies to key committers**). In the worst case, the project might depend on one particular person. This is obviously a risky situation for the project and its users as the key committers may leave which then brain drains the project, and demotivates other developers, as the case in Apache Slide (Wahyudin, Schatten, Mustofa, Biffl and Tjoa, 2006).

The dominance of the key committers may also reduce the opportunities of peripheral developers to contribute, which can be considered as a hostile action. This dependency has been considered as a major issue by Apache Software Foundation that should be comprehended by all new project initiatives under the Apache incubation process. Other typical risky situation that may threaten OSS project is the shift of market or the change of technology which causes project’s disorientation and consequently demotivates the developers to abandon the project (like Native XML Database).

2) User Community Liveness:

The second group in the OSS project community is the user who observes, downloads and then uses the software product for certain objectives (e.g. curiosity, work functions, and user needs). The level of software **use intensity** will be amplified when certain quality attributes of the software product **satisfy** the user value expectation. Compared to those of commercial software products, the users in OSS project are expected to be more active to provide **feedback** on the functionalities of the product release. Some studies (Fogel, 2005; Raymond, 2003) reported that most of the bug reports and the feature requests came from the users community, which were then responded by the developer community by submitting patches or new features. Eventually, these practices caused rapid changes into the code and documentation. However the user expectations may change over the time due to the technology evolution, needs of reorientation or some other reason. Eventually these changes imply the **shift of trend and demand of the market** which consequently impact the development process. Hence, it is fair to say that the user community has significant impact on the OSS project community liveness in the whole and on the quality of the software product released.

3) Product Quality

The typical characteristics of OSS such as being open-code based and no formal project management have raised some debates about the quality of the released products. However a survey from BCG (Group, 2002) suggested that open source community is mostly comprised of highly skilled IT professionals who have, on the average, over 10 years of programming experience and it is not exaggerated to assume that these people are well knowledgeable to produce a good quality code which is contrary to popular belief about hackers.

Recent study in OSS quality (Aberdour, 2007) suggests several software engineering quality model that typically practiced in OSS project community such as peer review to assess whether a contribution merits from developer acceptance into the code base. The bazaar style of OSS development facilitates rapid releases which make the implementation of peer review. The existence of quick response to reviewers comments and code keeps the contributor involved and interested (Raymond, 2003). As in a large project such as Apache HTTP Server, peer review is practiced not only for assessing the quality of contributed code but also applied for a new idea/solutions submitted to the developer community which need to be discussed, and reviewed before being planned for development.

The second typical quality practices are people management in reporting, reviewing, debugging and resolving issues. Abernour (2007) advocates on this practice to include establishing an effective environment and culture which is as important as system design. This means there should be a pre-defined coordination mechanism (Fielding, 1999), conflict management (such as voting) (O'Reilly, 1999), encouraging innovation and creativity (Lawrie and Gacek, 2002), and affectionate attention from the community (Lerner and Triole, 2002).

The third and the most prominent quality practice is bug tracking activity. In a traditional project, a bug tracking is similar to an inspection which is effective but also expensive quality assurance. A study from Ximian Evolution (German and Mockus, 2003) disclosed that after a product release, the user and larger part of the OSS community typically shift their roles in reporting, reviewing and resolving bugs/issues. These practices of software engineering quality model influence the community liveness by encouraging project participants to be involved and be motivated to keep contributing, resulting in a product that extends rapidly and reaches high quality. Here we conclude that *only a healthy community can produce high quality software*.

B. Project Health Indicators

In the previous section many different parameters had been taken into consideration to get an impression of the status of a project which is actually not an easy task. This is particularly problematic if a large number of projects needs to be monitored. Based on the described success factors, there are some indicators that experts routinely use to assess an open source project, such as:

- *Open issues, service delays*. Bugs and issues are listed in the bug-tracking system, but the fixes are not done in appropriate time.
- *Proportions*. We calculate the proportions of activity in the community, e.g.: the volume of mailing list postings, bugs status changes per time slot, updates in the SVN; and use these metrics to compare different projects to try to learn what ”healthy” relationships are.
- *Communication and use intensity*. If a project has a healthy community, there is indication of strong relationship between some measures such as the number of download compared to mailing list postings and the active developer interaction in (different) mailing lists.

In a distributed project, a set of certain software engineering tools and standards have been established to support open source projects. The tools are used where activities of developers can be interpreted as process *events* (time stamped date points) data. Typical important tools are: source code repositories, systems documentation (source, user), bug tracking, mailing lists, software forum (web, newsgroup), and Wiki content management. These tools provide informative but scattered pools of data during a project’s life-cycle. Obviously, developers use these tools and data for coordination, communication, and configuration management.

We propose an approach by unifying the data coming from different systems into a coherent format for analysis. We expect not only data for historical analysis but also for daily or weekly monitoring and analysis. If this is achieved, it is possible to monitor the status of the project’s health regularly and receive early warning signals, if bad smells occur. For analysis it is necessary to collect, filter, and correlate these data elements. While a human expert has to do the analysis by looking into the different systems, only little tool support is readily available for automatic and continuous ”real time” analysis of project status.

In today’s practices, health indicators and healthy community of OSS projects have increasingly become important issues. Just to give an example, the Apache Incubator[4] defines that a new project initiative may pass the incubation process by fulfilling some requirements: *the project must have a healthy community indicated by an active collaborative works within the community and it consists of diverse core developers*. For the diversity of core developers’ measurement, we can quantify the number of

[4] http://incubator.apache.org/ (accessed at 01/02/2007)

independent core developers based on their background profile. The diversity is important because: (a) it guarantees a sustainable development, as the project will be less dependent on a single developer, (b) it brings variety of competencies to enrich the quality. However our interview with OSS expert suggests that this indicator is best to be obtained manually by retrieving each committer personal data and analyzing the project profile. Active collaborative works are indicated by several health indicators such as the coordination activities, conflict resolutions (number of voting), intensity of usage, bug service delays, and the proportion of the developer contribution to the project repositories. In this paper we focused on the last two health indicators.

The *service delay*, in a commercial project, is the time interval to respond a service request from a customer and time to fulfill the service. While in OSS project, we define the service delay as a function of time to respond and time to resolve an issue/bug. The bug/issue service delay is important as most of the activities in OSS project are derived from bug or issue report; eventually a project which has slow response and resolution time will face problems such as user dissatisfaction and bottleneck in the development process. The second health indicator is the proportion of the developer contribution. These indicators are very important to obtain an outlook of software creation performance and the current developers' motivation state. Both of the indicators are derived from the aggregation of metrics, which at the lowest level the metrics are obtained by mining the project repositories as illustrated in the Figure 2.

C. Stakeholder Value Proposition for OSS Project Health Monitoring

The stakeholders in OSS project are represented by each individual in the community. To better understand their expectation about project monitoring we need to elicit their values; starting by defining win conditions and their key measurement and then select which requirements meet the win conditions (Biffl, Aurum, Boehm, Erdogmus and Gruenbacher, 2005). In this paper we focus on eliciting the value of project manager as the most prominent user in monitoring day to day a large (employing more than twenty active developers as listed in each project's web site) web-engineering project.

Based on our observation from several OSS projects, we found that a project is typically led by a group of key committers or steering board. Just to give an example, from projects under Apache Foundation umbrella, we found a form of project leading team called as *Project Management Committee* (PMC). The PMC members are originally elected from active committers based on their merits within a project and then appointed by the Foundation Board to sit in the PMC (later we refer this individual member as the project manager).

According to the Apache Software Foundation[5], the main roles of the PMC are to *ensure that all legal issues are addressed and the procedure is followed, the alleviation of any bottlenecks and conflicts, the overall technical success of the project and that each and every release is the product of the community as a whole* . PMC is also responsible to give strategic decision, to further the long term development and the health of the

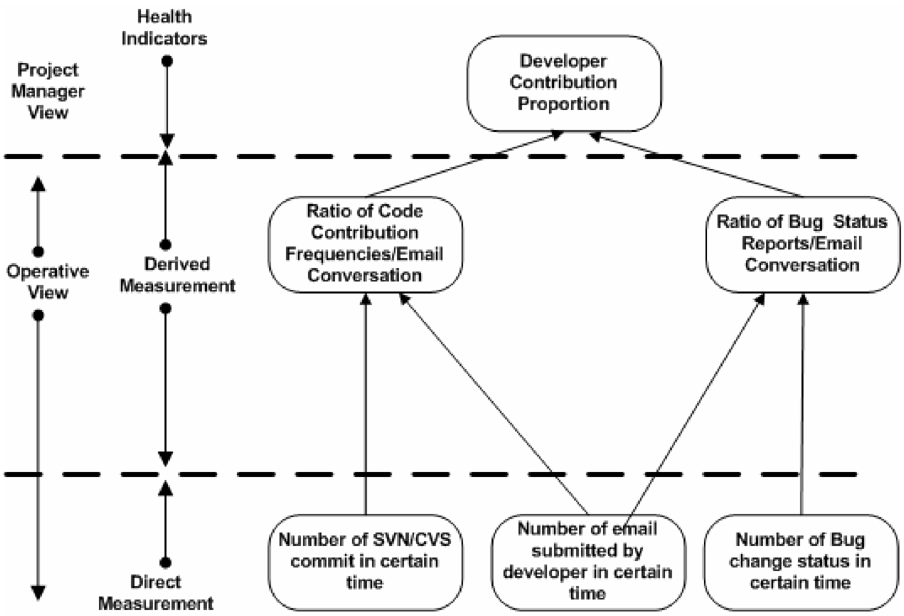
[5] http://www.apache.org (accessed at 28/01/2007)

community as a whole, and to ensure that balanced and wide-scale peer review and collaboration do happen.

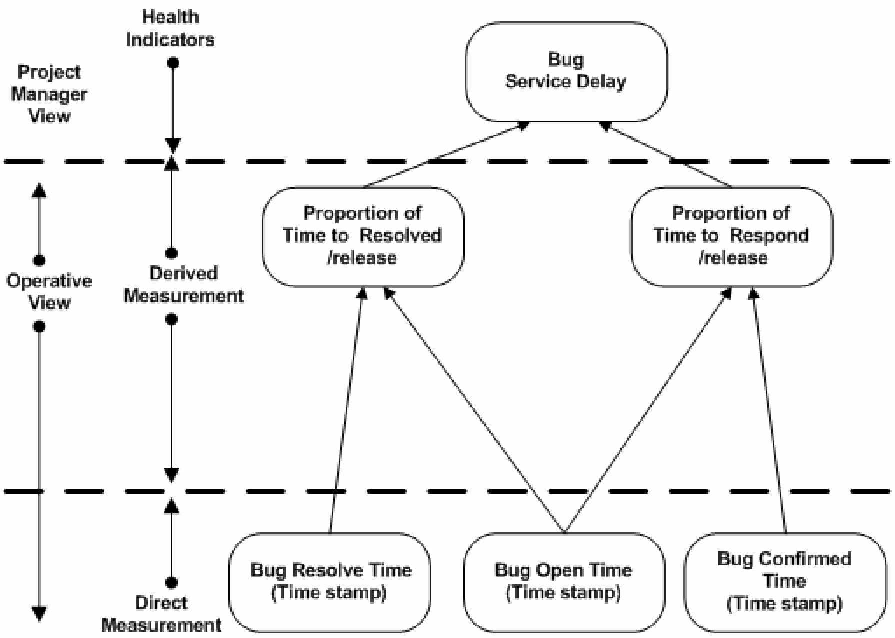
These roles depict the need to monitor the health of the community in timely fashion, and to quickly respond appropriately against certain status during project execution. Hence the common win conditions of project manager in monitoring OSS project are:

- *To retrieve comprehensive indicators which indicate the actual status of the project performance*. During execution, a project may produce a significant number of data, artifacts and project information which can be further processed as indicators. However due limitation of observers in OSS project as they are not a full timer, compacting the indicators with respect to quality of the enclosed information is a necessity. The key measures of this win condition probably to have a small set of indicators based on observer's own priority selection. Just to give an example, from the IT system monitoring, our interview with a group of system administrators results that the group daily monitors eight to ten indicators, as to have more may overwhelm and mislead the observer's analysis.
- *Availability of layered performance metrics*. This second condition is the result of the first one, as in a project we may start to monitor a higher level indicator and then go deeper for better understanding the nature of the development process or tracking back the symptoms of illness. We suggest to divide the metrics into three layers: (1) *the direct metrics*, which is the lowest layer consisting of metrics directly obtained from the project repositories such as number of developer email contributions, time stamps when a bug is resolved, the number of bug residing in the bug database etc. (2) *the indirect metrics*, which is the aggregation of several direct metrics such as the average time to resolve bugs in certain time, average number of developer email contribution in one semester, etc. (3) *the indicators*, which is the highest level and provides comprehensive view of certain status of the project, e.g. the service delay within a release depicted as a function of the average time to respond and average time to resolve of bugs for the release. Figure 2 illustrates the examples of retrieving health indicators from a hierarchy of metrics.
- *Early availability of time relevant project information status*. The third win condition is to have quality information in a short time, as the project indicators reflect the current status which is not merely historical data. Time relevant information will give more accurate indication of the project, and if the project is considered unhealthy, appropriate treatments can be applied to change the project into healthy one before it is too late or getting worse. To get valid information status the project manager should set a retrieval time constraints for each indicator he wants to monitor: i.e. within hours, days, or months.

The stakeholder values vary based on the stakeholder roles, domain of the project applications and the project scales. The value elicitation will define the selection of to-be-monitored health indicators to assess the project health status.



(a) Developer Contribution Proportion



(b) Bug Service Delay

Fig. 2. Measuring OSS Project Health Indicators. The measurement selection for monitoring project status, depends on the stakeholder values. The Operative level measure direct (low level) metrics from project repositories which are then transformed into higher level metrics. Project manager analysis the health indicators as the results of correlation of these aggregated metrics

IV. EVALUATION PROCESS

This section describes the evaluation process of the health indicator concepts and the initial empirical result from some projects under the Apache umbrella.

We apply the proposed health indicators to four cases of large-matured Apache web engineering projects indicated by more than 20 contributors (core and peripheral developers) per project which three times outsized the average number of developer in most of OSS projects (Manenti, Comino and Parisi, 2005) 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/ HTTPD 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*. As described in Section III these indicators are very worth noted by a project manager to assure that a project is still actively running and both indicators are simple to be evaluated.

A. The Origin of The Four Projects

- **Apache Tomcat**[6] 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 the Java Community Process in 1999. The first Apache release was version 3.0. Since then, multiple volunteers from Sun and numerous other organizations have contributed to the product. Currently Tomcat has several major releases, employs 17 active committers and more than 50 emeritus committers. In 2005, Tomcat became its own top-level Apache project and powered numerous industries and organizations such as Wall Mart and General Motors. A survey by `TheServerSide.com`[7] pointed Tomcat as one among the market leaders in its application domain.
- **Apache HTTP Server**[8] is an effort to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The development started in 1994 when Brian Behlendorf and a number of users for internet servers which are developed by the National Center for Supercomputer Applications (NCSA) encountered the increasing frustration in getting NCSA to respond to their suggestion. They decided to collaborate and integrate patches to the NCSA server software. In August 1995, the group released Apache 0.8. Since then the product was called as Apache HTTPD. Later it is well-known as Apache HTTP server and has been the most popular web server on the Internet since April 1996. The HTTP server project employs 56 core contributors and hundreds of peripheral contributors. The project latest release is the version 2.2.4.

  Apache Tomcat and HTTP Server are considered successful large projects. Hence by examining the dynamics of both project communities, we improve our knowledge about the indication of a healthy community.

---

[6] http://tomcat.apache.org/ (accessed at 20/02/2007)
[7] http://www.theserverside.com/ (accessed at 10/01/2007)
[8] http://httpd.apache.org/ (accessed at 20/02/2007)

- **Apache Xindice**[9] 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. Xindice is the continuation of the project that used to be called the **dbXML Core**. The dbXML's source code was donated to the Apache Software Foundation in December 2001. During the development, the developer community consists of 8 active committers, 7 inactive and emeritus committers, and 19 contributors. Xindice has its first stable release of version 1.0 around March 2002 and continues with several milestone releases, such as version 1.1b4 at 8 April 2004.
- **Apache Slide**[10] is part of the Apache Jakarta project. Slide is a content repository which can serve as a basis for a content management system / framework and other purposes. The original Slide codebase (Slide 0.7) was donated by Intalio Inc during May 2000. Slide has reached its maturity after release 1.x and 2.x. The project employs 14 active committers, 14 inactive/emeritus committers, 20 contributors and 3 project sponsors. However after its 2.1 release (at 12/26/2004), the project seemed to be disposed, although we still recorded some activities in the developer mailing list. Our interview with OSS expert indicates that Apache Xindice and Slide are in difficulties. This case can be taken as a fine comparison to successful ones and reveal the symptom of illness of a project.

B. Evaluation Process of the Proposed Health Indicators

In an OSS project, the developer mailing list is the main collaborative communication tool, where everyone who wants to participate in the project development can observe or join in. In Apache projects, the email archives commonly consist of three major contents: *the notification of developer commits to the SVN, notification of bug status report (the change of bug state) as a developer may work on something for the bug, and development-related short messages/email conversation* i.e. problem reports, solution recommendation, polling for opinions, and technical discussion. The proportion of the developer contribution can be measured from the ratio between number of email conversation, SVN/CVS Commits and bug status reports. We addressed two questions for measuring the developer contribution: (1.) *what is the ideal proportion of emails submitted to the mailing list*? (2.) *Are there any correlations among developers' contribution components*?

To answer the above mentioned questions, we need to collect numerical data about some existing OSS projects. For the purpose, we developed a tool for mining the web based developers mailing list (hosted by Mailinglist ARChive, MARC) of each selected project. The tool uses the approach of a wrapper which retrieves project data web pages and then parses them to extract the information required. It retrieves the emails data (sender, subject, thread, time stamp) based on some given time interval as the input parameter. The whole outcome is represented as XML and then using XSLT is further transformed to finally result in performance metrics, such as number of emails, number of CVS commits and number of bug status changes. For each project we select and examine 38 months of projects' life time with at least one major stable release. To support viewing

[9] http://xml.apache.org/xindice/ (accessed at 20/02/2007)
[10] http://jakarta.apache.org/slide/ (accessed at 20/02/2007)

the dynamics of the projects, these emails are then grouped into monthly archives and the proportion are calculated. We illustrate the ratio of developers' contribution over the time and compare the result between the successful projects and the challenged ones. Later, we choose Apache HTTPD as the role model of a healthy project and employs linear regression to figure out the correlation between email conversation, code contribution and bug report status.

Additionally a healthy project should employ proven bug management practices to reduce the service delay and offer fast bug response. In an OSS project, the community voluntarily plays significant role in the bug tracking, and resolve the financial barrier as in traditional project. Here we address 3 questions for assessing bug management in a project: (1.*) Is there any appropriate bug reporting and monitoring in place*? (2.) *Is there any appropriate rating of bugs*? and (3.) *What is the distribution of response time and closure time of bug reports*?

Apache Project has centralized it bugs tracking within repositories managed by GNATS and later moved to Bugzilla. The Bugzilla offers more features like transaction logs (history) and search facility either simple or advanced search on descriptive information of bugs. This makes the dynamics of bugs are relatively easy to trace. It is not surprising that Bugzilla became very popular and widely used by 550 projects or companies[11]. This indicates that each Apache project being evaluated implements appropriate bug tracking tools.

In order to evaluate the performance of projects based on bug's statistics for one stable release of each project, we retrieve each project's bug reports, measure quantitative data items, and depict the result within one software major release. Unfortunately, some data needs pre-processing due to inappropriate or illegal values. The pre-processing steps involve: removing records indicated as "duplicated" in the resolution field and excluding records containing invalid date (either in the open-date or change-date).

We retrieved and examined CVS logs from the bug database of Apache Tomcat 5, HTTP Server 2.0, Xindice and Slide. Furthermore, in measuring the distribution of response time and closure time of bug reports, we formulate the calculation using the criteria: (a) response time is the length of time interval between open date and last change date for the bugs having status field set to "NEW", which means the bug is confirmed and accepted by the community for further processing, and (b) closure time is time to resolve a bug, calculated by measuring the length of time interval between open date and last change for bugs having status "RESOLVED", which means the bug is already went through the development processes.

---

[11] http://www.bugzilla.org/installation-list/ (accessed at 25/02/2007)

V. INITIAL EMPIRICAL RESULT

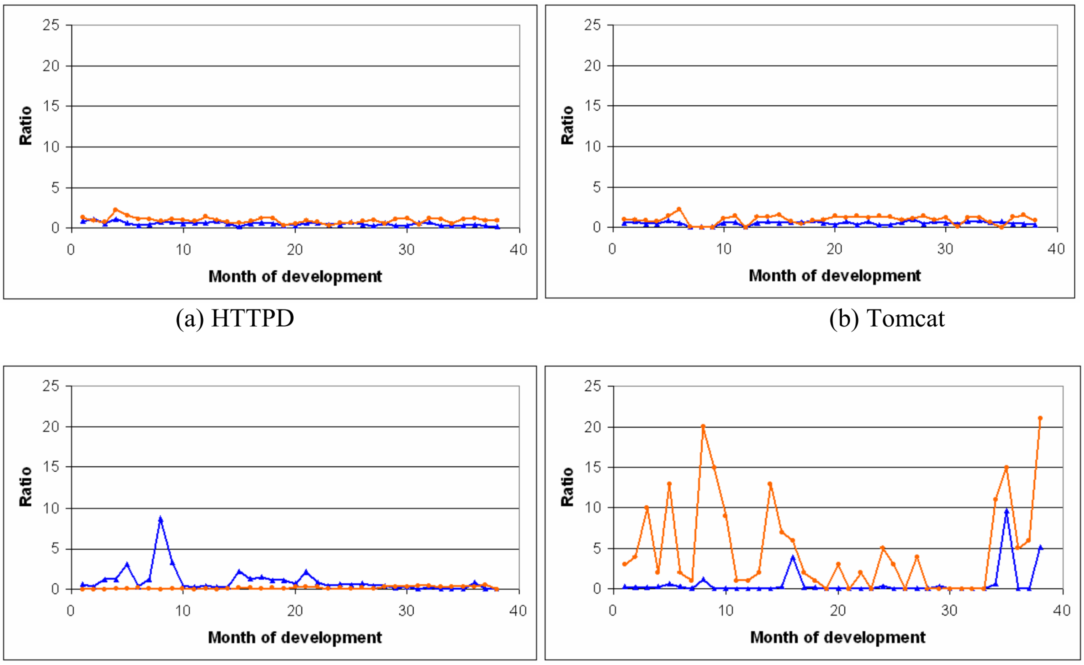
This section describes the initial empirical result which is composed of data collection from the four Apache projects and data analysis.

A. The Developer Contribution Level

A study (Wahyudin et al., 2006) found that in the two successful projects a positive trend line of developers' email contributions during project life time exists (see Figure 5 on the appendix). On the contrary, the two challenged projects reveals dying periods as revealed in the diminishing absolute number of developer contribution depicted in the Figure 6 (of the appendix). In this paper we look further the cause of the illness symptoms and what a healthy relation should look like between components in the developer contributions.

1) *Developers' Contribution Patterns in The Four Reviewed Projects*:

To better understand the developer contribution patterns within different types of project, we examine the retrieved data set, and calculate the ratio between code contribution (number of CVS commits), and reports of bug status (number of bug status change notification) with the developer email conversations. We argue that in a healthy community the project should exhibit more proportional ratio among these metrics, i.e. every CVS commit ideally should be followed up by the developer discussion in the mailing list before inserted into the body of code. We found the both challenged projects exhibit more fluctuation and higher ratios as illustrated in Figure 3. It means, the developer community retrieved more notification of code contributions and bug reports but responded less in the mailinglist. This situation may indicate illness symptoms. We considered some of the illnesses are the facts that the developer community pays less attention to project status' changes; the project employs small proportion of active developers which also signify developer demotivation which further needs to be investigated by experts in OSS Community. On the contrary, both the HTTPD and Tomcat signify more reasonable proportion in the ratio of developer contribution. This can be interpreted that most of the developer contributions triggered some responses from the developer community.



(a) HTTPD (b) Tomcat

(c) Slide (d) Xindice

Fig. 3. Developers' contribution patterns. In the healthy projects (3(a) and 3(b)) the number of code contributions per developer email and the number of bug status report per developer email tend to be proportional. On the contrary, the challenged projects (3(c) and 3(d)) show fluctuations which signify the imbalance between code contributions/developer email submissions and bug status reports/developer email submissions.

2) Email conversation as a function of CVS/SVN commits and Bug Status Changes

As described above, a successful project community exhibits more normal distribution of developer activities metrics during the development process. The next step, we want to find out the correlation among these metrics. For the purpose, we perform a regression model analysis. We use the data set retrieved from Apache HTTPD as the role model. We assume that the result will suggest the correlation of developer contribution metrics in a healthy project. The data set retrieved from HTTPD consist of 38 months of observation, with the total of 15781 email conversations (μ=415.2; δ=135.9), 14199 SVN/CVS commits (μ=373.6; δ =99.4), and 8757 bug status changes (μ=230.4; δ =106.1)[12]. To observe more relationships among the measures, we employ the regression linear analysis, and model the developer email contribution as a function:

$$Y = b_0 + b_1 X_1 + b_2 X_2 + \varepsilon \qquad (1)$$

Where Y is number of email conversation, X1 is number of CVS commits and X2 is number of bug status changes. The confidence interval of our test is 95%. The test consist of two phases: first we test the significance of the regression coefficients of our model to find out the impact of CVS commits and bug status changes to the number of email conversation, second we test the regression model itself to assess its validity.

[12] we use " μ" to represent monthly average and " δ" for its standard deviation

i. Regression Coefficients Significance Analysis. The statistics test for hypothesis of each coefficient regression is the Student's t-test which can be calculated as:

$$t_0 = \frac{\hat{b}_i}{\sqrt{\hat{\sigma}^2 C_{ii}}} \tag{2}$$

Where C_{ii} is the diagonal matrix element which correlated with $\hat{b}_i$

We calculate the model using SPSS and test the following hypothesis:

- H0: coefficient regression is not significant
- H1: coefficient regression is significant

The analysis will reject H_0 if:

$$|t_0| > t_{\alpha/2, n-2} \tag{3}$$

where $t_{table} = t_{0.025,36} = 1.960$

a) Constants.
The calculation results t_0 3.673 for the constant. Since 3.673 > 1.960 H_0 is rejected. This means that the constant has significant impact to the regression model

b) CVS Commits.
The calculation results t_0 4.715 for CVS Commits. As 4.715 > 1.960 H_0 is rejected. This means the Bug Status Changes has significant impact to the number of conversation email

c) Bug Status Changes.
The calculation results t_0 -0.537 for the Bug Status Changes. Since -0.537< 1.960 H_0 is not rejected.

This means although bug development is an important issue in OSS project, however the bug report status has less and does not show significant impact for triggering developer conversations in developer mailing list.

ii. Regression Model Significance Analysis.
We employ the *F-test* for analyzing the proposed model, which can be calculated as:

$$F_0 = \frac{MS_R}{MS_E} \tag{4}$$

We calculate the model using SPSS and test following hypothesis:

- H0: regression model is not significant
- H1: regression model is significant

The analysis will reject

$$F_0 > F_{\alpha, n-2} \tag{5}$$

where F_{table} or $F_{0.05,36}$= 4.11.

The calculation results F_0= 11.975. Since 11.975 > 4.11, H_0 is rejected. The test concludes that the model is valid and there is a linear relationship between Email conversations with CVS Commits.

The above results conceal that in a healthy project (such as HTTPD) the code contribution has significant impact to amplify the number of email conversations in the developer mailing list such that the ratio between these two variables are kept in proportional during the development process which signifies one healthy status. The bug status reports are also important to illustrate the project service throughput. However we suggest that monitoring the dynamics of the bug status report should be further correlated with other variables in the bug tracking activities such as the number of bug per reporter, bug response per reviewer, bug assignment per contributor, service delay (response and closure time), validation time etc, which we will cover in our future work as other types of project health indicator.

B. The (Bug) Service Delay

Based on the measurement scenario on bug service delay as mentioned in section IV, we present these following quantitative results. The first measurement is to see the distribution of the bugs' severity on each project. In the Apache projects, the bugs are categorized based on their severities related to security (critical) and fault (blocker), related to feature (major, minor, enhancement, normal) and related to cosmetic works (regression and trivial). Later, for further processing, the community put the development priority (P1 to P5) for each bug, where the P1 means the top priority and needs to be resolved as soon as possible.

We retrieved the bug data from Apache Tomcat 5 (2891 bugs), HTTPD 2 (3663 bugs), Xindice (152 bugs) and Slide (420 bugs). On the data, we examine the distribution of the bugs based on severity and priority to find the proportion of bug assignment. We observe also a situation that the distribution of the bugs based on severity is almost "normal" as illustrated in table I (see the figures in italic bold), in the sense that most of the bug reports coming from the user community are feature requests, functionality errors or decorative ones. Furthermore, from the tables we can see that although the user community reports bugs and considers the bugs of high severity (such as "blocker"), the priority assignment by developer does not always follow the "user needs". In other words, the developer does not always assign high severity (according to user) with high priority.

To measure the performance of bug service delay in a project, we measure the bug response time (*time to respond*) and the bug closure time (*time to resolve*). We categorized such service delay into several time interval scales as shown in the Table II. Figure 4 shows the distribution of bug response time (Tr) and bug closure time (Tc) from the four reviewed projects. From the figure it is obvious that Tomcat has the most responsive community, as 60% of the reported bugs are responded less than 7 days, 79% are responded within 100 days. On the other hand, Xindice exhibits poor performance as the majority of the bug reports (72%) were responded by the community in more than 100 days.

The successful projects provide faster service as their bug closure times are shorter compared to those of Xindice and Slide. The results are about 46% of the bugs in Tomcat and 59 % of the bugs in HTTPD are resolved within 100 days, while most of the bugs reported for the challenged projects are resolved within or more than a year. However our result also found intriguing fact that more than 20% of the bugs were resolved more than a year in HTTPD and Tomcat. We investigated this issue by measuring the absolute number of the resolved bugs and correlated with the assigned priority. For HTTPD, among the 999 resolved bugs, 24% are latent since they were resolved more than a year; 88% of these latent bugs are categorized as lower priority (P3, P4, P5) and 45% are considered as cosmetic work or minor feature error. For Tomcat, from 2063 resolved bugs, 27% are latent bugs with only 9% of top priority bugs (P1 and P2), and less than 15% are severe bugs (critical and blocker). We conclude that in both successful projects, the community offers faster response to a bug with higher priority, and tends to delay the less important ones.

VI. DISCUSSION

For many OSS developers, challenge is what really motivates them and it makes the project more active. Once they are drawn to a problem, they feel that they could create a better solution themselves, rather than using the existing ones. Over the time, the projects may evolve and so does the motivation of the developer.

Our study in OSS project community health revealed two extreme trend lines in the developer participation: (1) *lively developers' activity as shown by Tomcat and HTTPD*, (2) *non responsive or dying developers' community*, which we observed in Xindice and Slide.

The developer community in the challenged projects shows less normal proportion of developers' contribution metrics (such as CVS commit, bug change status and email conversation). This condition is considered to signify situation in which the community is less responsive to the status changes within the project. In a healthy project such as HTTPD we found a correlation between CVS commits with the email conversations in the developer's mailing list. This indicates that, in general, a contribution of code or document into project repositories may attract other developers to interactively involve in discussions and reviews.

Xindice obviously reveals a very fluctuating proportion of developer contribution (see Figure 3(d), which means that there are significant imbalances between the contribution and the response from the developers' community). Furthermore based on the absolute number of the developers' contribution, Xindice indicates a"dying" project, which can be seen in Figure 6. In its last 33 months, the developers' contribution levels were low compared to its first 5 months under observation. When we discussed with some experts, according to them, the reasons behind the condition are: (a) *Xindice has been abandoned by its key developer*, which also proves that diversity of core developers is important (b) *the changes of market demand and technology trend* caused the developers' community to recognize that proposed plan and the results not to evolve expected.

TABLE I
BUG PRIORITY DISTRIBUTION PER SEVERITY

(a) Tomcat 5

Severity	% Priority				
	P1	P2	P3	P4	P5
blocker	0.7	0.8	2.1	0.0	0.0
critical	1.4	2.0	4.1	0.1	0.1
enhancement	0.1	5.3	5.6	0.2	0.7
major	0.8	5.7	9.7	0.1	0.1
minor	0.1	1.8	3.8	0.4	0.3
normal	0.7	***26.7***	***24.4***	0.5	0.3
regression	0.0	0.2	0.0	0.0	0.0
trivial	0.0	0.7	0.1	0.2	0.5

(b) HTTPD 2

Severity	% Priority				
	P1	P2	P3	P4	P5
blocker	0.7	0.9	4.3	0.1	0.1
critical	1.2	1.9	6.8	0.1	0.1
enhancement	0.1	2.7	6.5	0.1	0.5
major	1.0	4.3	9.3	0.0	0.0
minor	0.0	1.7	6.6	0.2	0.5
normal	0.9	***17.3***	***30.3***	0.2	0.2
regression	0.0	0.5	0.0	0.0	0.0
trivial	0.1	0.5	0.1	0.1	0.2

(c) Slide

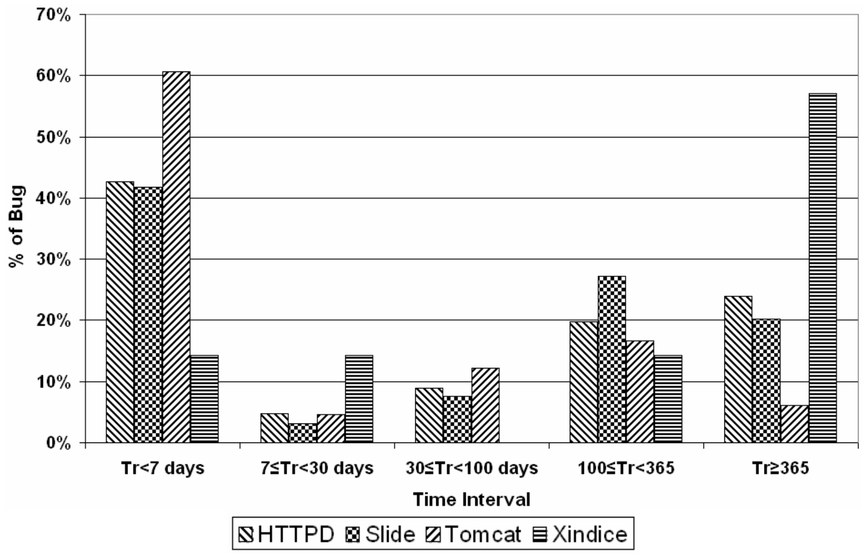
Severity	% Priority				
	P1	P2	P3	P4	P5
blocker	0.5	0.2	1.7	0.0	0.0
critical	1.0	2.6	4.5	0.0	0.0
enhancement	0.0	1.9	7.4	0.0	0.2
major	1.2	5.7	5.5	0.0	0.0
minor	0.0	0.5	6.9	0.2	0.0
normal	0.5	***17.6***	***41.7***	0.0	0.0
trivial	0.0	0.2	0.0	0.0	0.0

(d) Xindice

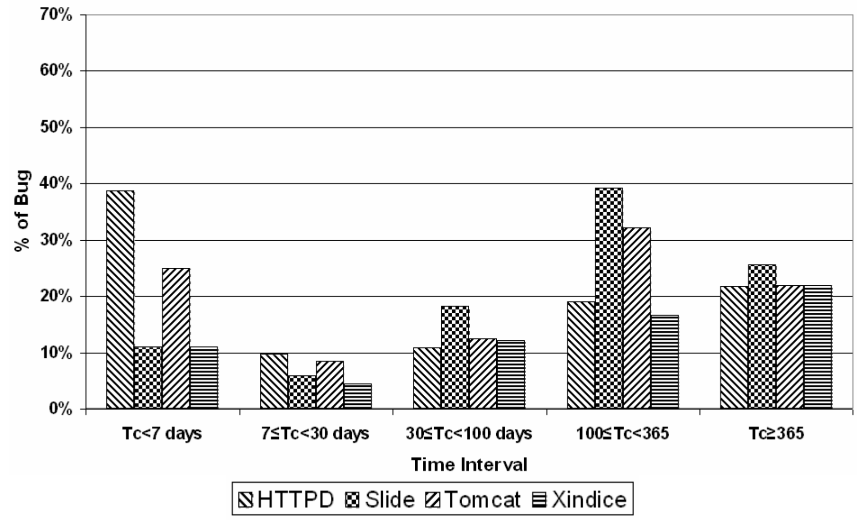
Severity	% Priority				
	P1	P2	P3	P4	P5
blocker	0.7	1.3	3.3	0.0	0.0
critical	0.7	0.7	7.2	0.0	0.0
enhancement	0.0	0.7	5.9	0.0	0.7
major	0.7	1.3	***14.5***	0.0	0.0
minor	0.0	0.7	4.6	0.0	0.0
normal	0.0	7.2	***48.0***	0.7	0.0
trivial	0.0	0.7	0.7	0.0	0.0

TABLE II INTERVAL SCALES FOR BUG TIME TO RESPOND AND TIME TO RESOLVE.

	Interval Scales For Bug Service Delay				
	Less than a week	in weeks	less than 100 days	less than a year	in years
Range(Days)	0<=t<7	7<=t<30	30<=t<100	100<=t< 365	days t>365



(a) Bug response time



(b) Bug closure time

Fig. 4. Bug Service Time distribution for reviewed projects. A Healthy project should be more responsive to bug report and offer faster bug resolution time

Slide has a different story as shown in Figure 3(c). The project exhibits proportional developer contribution, however based on the result of measuring the absolute number, on the beginning Slide was very promising, but since October 2004 the project was hit by catastrophic illness indicated by the decrement of its developer contributions (Figure 6). The experts participating in the Slide project mentioned that Slide was once in a dormant state. The project woke up after a talented expert got involved in November 2003, and significantly contributed for the peak performance of Slide. However after several months, he decided to leave the project, leading to the collapse of Slide. A study on this case (Wahyudin et al., 2006) validated this statement by measuring this experts' mail conversation and suggest the fluctuation in developers' mail follows the dynamics in

”The Expert” mails. Around March 2004, as the expert decided to leave the project, Slide developers’ mailing list still showed notable activity for several months, before their number finally collapsed. Our observation in February 2007 disclosed although there are still some developer activities, the level is relatively low compared to the zenith period when ”the Expert” was still actively involved in the project.

Our study also reveals that a healthy community will be more responsive and more eager to resolve issues or bugs introduced to them. However the empirical result of bug severity distributions argued that the more severe bug does not always mean to be of higher priority, since most of the top priority bugs are the normal ones as illustrated in Figure I. It is likely that the response time is affected also by priority set by developers due to some consideration. For example, a blocker might be given lower priority when it occurs very rarely and it is planned not to be resolved in the current release. This practice exhibits value tradeoffs between the users who report the bug, assign the bug severity and expecting quick response, with the community who assess, assign, develop and resolve the bug.

VII. CONCLUSIONS AND FUTURE WORK

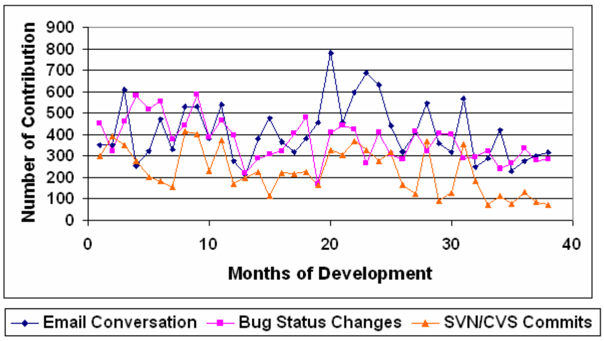
Managing and monitoring health in open source software (OSS) projects is a complex challenge, due to the typical characteristics of OSS development model. Important indicators such as activity of developers and performance of bugs’ management are easier to measure as they have become part of the development nature itself. However, for the comprehensive determination of health measurement one has to consider other indicators which needs be formulated and further explored.

Some of these to-be-formulated indicators are“hidden” behind the development process. Hence, effectively, it is the core stakeholder who should make the decision about which indicators should be employed, based on the projects initial needs. In principle our approach is applicable for commercial web-engineering project, however further work should investigate the commercial project structure and the culture within the team development to reveal appropriate health indicators.

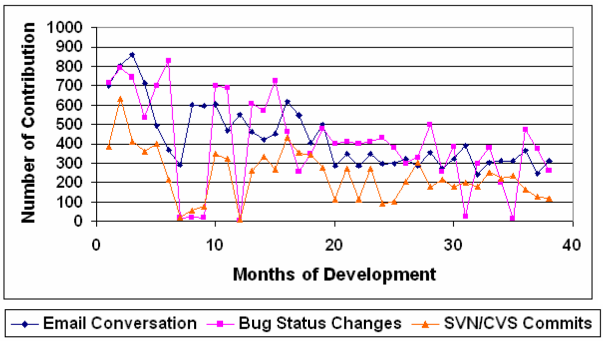
The investigation of important health indicators is just the beginning. The next step is the development of assessment methods that allow observers to get semi-quantitative measures of project health itself. This will help people to learn how these processes work in-depth, allow to enhance cooperation and give monitoring and ”early-warning” capabilities to the project stakeholders. Nevertheless more works should be done to define relevant dynamics indicators, empirical rules and measurement metrics for an on going OSS project quality and project community assessment.

APPENDIX I
DEVELOPER CONTRIBUTION ABSOLUTE NUMBER

Figure 5 and Figure 6 exhibit the absolute number of developer contribution within 38 months of development of the reviewed projects. The developers' contribution patterns are distinguished into three line categories: the code contribution (number of SVN/CVS commit), the bug status report (number of bug status changes in the bugzilla), and the developer email conversation (number of email submitted into the developer mailing list). In healthy project such as Apache HTTPD and Tomcat (see Figure 5), the developer communities are very dynamics with high level of contribution. In contrary, both the challenged projects (see Figure 6) face diminishing developer activity, especially in Xindice which can be considered as a dying project.

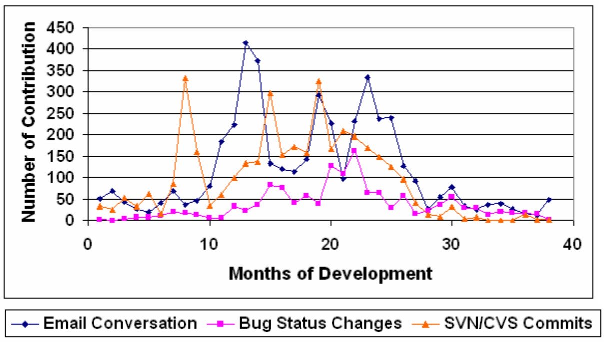


(a) HTTPD

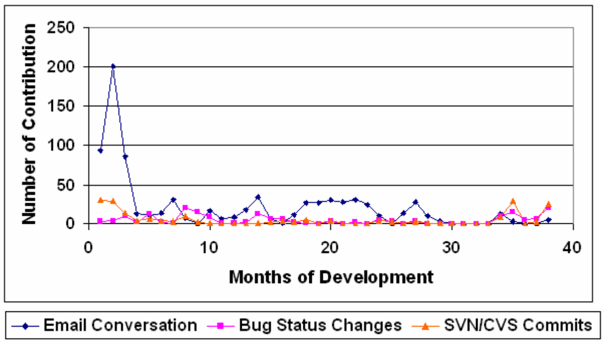


(b) Tomcat

Fig. 5. Absolute number of developer contribution in the successful OSS projects (HTTPD and Tomcat)



(a) Slide



(b) Xindice

Fig. 6. Absolute number of developer contribution in the challenged OSS projects (Xindice and Slide)

ACKNOWLEDGEMENT

This paper is part of larger research called Monitoring Distributed Software System Development and Operation. The objective is to define more effective and efficient methodology for monitoring distributed running system and distributed software project management in order to satisfy various stakeholder' needs. This project has been partly supported by The Technology-Grant-South-East-Asia No. 124-2/BAMO/2005 and The North South Dialogue Scholarship Program EZA-Project 894/05, financed by ASIA-Uninet in cooperation with the Austrian Council for Research and Technology Development

REFERENCES

Aberdour, M. (2007). Achieving quality in open-source software, *IEEE Software* **24**, **Issue**: 1: 58–64.

Biffl, S., Aurum, A., Boehm, B., Erdogmus, H. and Gruenbacher, P. (eds) (2005). *Value-Based Software Engineering*, Springer Verlag, ISBN:3-540-25993-7.

Capiluppi, A., Lago, P. and Morisio, M. (2003). Characteristics of open source projects., *Proceeding of the 7th European Conf. Software Maintenance and Reengineering (CSMR 03),IEEE CS Press, p.317-330.*

Chen, K., Schach, S. R., Yu, L., Offutt, J. and Heller, G. Z. (2004). Opensource change logs, *Empirical Software Engineering* **Volume 9, Number 3 / September, 2004**: 147–262.

Crowston, K., Annabi, H. and Howison, J. (2003). Defining open source software project success, *Proceeding Twenty-Fourth International Conference on Information Systems*.

Fielding, R. T. (1999). Shared leadership in the apache project, *Communication ACM* **42: 4243**.

Fogel, K. (2005). *Producing Open Source Software: How to Run a Successful Free Software Project*, O'Reilly. **URL**: *http://producingoss.com/*

Gacek, C. and Arief, B. (2004). The many meanings of open source, *IEEE Software* 21(1): 34–40.

German, D. M. (2004). Mining cvs repositories, the softchange experience, *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR2004).*

German, D. and Mockus, A. (2003). Automating the measurement of open source projects., *ICSE '03 Workshop on Open Source Software Engineering, Portland, Oregon, May*.

Ginige, A., Murugesan, S. and Hansen, S. (2001). Web engineering: A new discipline for development of web-based systems, *Lecture Notes in Computer Science, Springer Berlin / Heidelberg* .

Boston Consulting Group (2002). Hacker survey. **URL**: *http://www.osdl.com/bcg/*

Keil, M., Smith, H., S.Pawlowski and Jin., L. (2004). "why didnt somebody tell me?" climate, information asymmetry, and bad news about troubled projects, *ACM SIGMIS Database* **32**(2): 65–84.

Krishnamurthy, S. (2002). Cave or community?: An empirical examination of 100 mature open source projects, *First Monday* .

Lawrie, T. and Gacek, C. (2002). Issues of dependability in open source software development, *SIGSOFT Software Engineering Notes* **27**(3): 34–37.

Lerner, J. and Triole, J. (2002). The simple economics of open source, *Journal of Industrial Economics* **52**: 197–234.

Manenti, F., Comino, S. and Parisi, M. (2005). From planning to mature: on the determinants of open source take-off, Discusion Paper, Universita Degli Studi Di Trento .

Mockus, A., Fielding, R. and Herbsleb, J. (2002). Two case studies of open source software development: Apache and mozilla, *ACM Transactions on Software Engineering and Methodology* **Volume 11 , Issue 3**.

O'Reilly, T. (1999). Lesson from open source software, COMMUNICATIONS OF THE ACM *April* **Vol. 42, No. 4**.

Phillips, J. (2004). *IT Project Management: On Track from Start to Finish, Second Edition*, McGraw-Hill Osborne Media.

Raymond, E. S. (2003). The cathedral and the bazaar. **URL**: *http://www.catb.org/esr/writings/cathedral-bazaar/cathedralbazaar/*

Royce, W. (2000). *Software Project Management: A Unified Framework*, Pearson Education.

Schatten, A., Tjoa, A., Andjomshoa, A., and Shafazand, M. (2001). Conference invited spech: Building a web-based open source tool to enhance project management, monitoring, and collaboration in scientific projects., *In Proceedings of the third International Conference on Information Integration, Web-Applications and Services*.

Schewe, K.-D. and Thalheim, B. (2005). The co-design approach to web information systems development, *Journal of Web Information System* **1 : Issue 1**: 5–14.

Thai, J., Pekilis, B., Lau, A. and Seviora, R. (2001). Aspect-oriented implementation of software health, *Proceeding Eighth Asia-Pacific Software Engineering Conference (APSEC'01).*

Valverde, S., Theraulaz, G., Gautrais, J., Fourcassi, V. and Sol, R. (March-April 2006). Self-organization patterns in wasp and open source communities,, *IEEE Intelligent Systems*, **Volume: 21, Issue: 2**: 36– 40.

Wahyudin, D., Schatten, A., Mustofa, K., Biffl, S. and Tjoa, A. M. (2006). Introducing "health" perspective in open source web-enginerring software projects based on project data analysis, *The Eighth International Conference on Information Integration and Web-based Applications Services.*

Dindin Wahyudin is a PhD student at the Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria. His research interest are value based software engineering, software project management, empirical software engineering, software economics and human behavior, and monitoring distributed software system and development.

Khabib Mustofa got a PhD from the Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria. His research interests are web technology including web services, information integration and the semantic web especially on semantic annotation.

Alexander Schatten is Researcher and University Assistant at the Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria. His research interests are applied internet technologies, project management, software engineering, and natural computation.

Stefan Biffl is an Associate Professor of software engineering at the Institute of Software Technology and Interactive Systems, Vienna University of Technology, Austria. His research interest are value-based software engineering, empirical software engineering, collaboration in software engineering, software process and product improvement, with application focus to Event-based systems for monitoring and coordination of heterogeneous software-intensive systems (http://qse.ifs.tuwien.ac.at/~biffl).

A Min Tjoa is a Full Professor of computer science at the Institute of software Technology and Interactive Systems, Vienna University of Technology, Austria. His research interest are data warehousing, data mining, semantic web, software engineering, information system, knowledge management, e-commerce, open source software development, EU framework projects, and information system for people with special needs.