Event-Based Monitoring of Open Source Software Projects

Dindin Wahyudin, A Min Tjoa
Institute of Software Technology and Interactive Systems
Vienna University of Technology, Favoritenstrasse 9-11/188, A-1040, Vienna, Austria
{dindin, tjoa}@ifs.tuwien.ac.at

Abstract

Project management traditionally has a strong focus on human reporting that fits well a tightly coupled form of organization to ensure the quality of project reporting. For loosely coupled forms of organization, such as open source systems (OSS) development projects, there are very few approaches to ensure the quality of project reporting; a promising approach can be to augment human reporting with data analysis based on the communication and state changes in an OSS project.

In this paper we propose a concept and an initial measurement approach for event-based monitoring of OSS projects to better understand the actual benefit of tool-supported gathering, correlating and analyzing processes event data from the OSS community as a supplement for traditional software project monitoring data collection. We report on an empirical feasibility study investigating success and risk indicators of five OSS projects listed in the Apache Incubator.

Keywords—Software Project Management, System and Process Monitoring, Event-Based System, Event-Based Project Monitoring; Open Source Software Project.

1. Introduction

Successful open source products, such as the Apache[1] web server, have obtained a significant role as an alternative business solution and have enjoyed industry-wide adoption. The *Apache Software Foundation* (ASF) proposes the *Incubator System*[2] for newly-added projects under its umbrella. *ASF* is mainly interested to invest in projects that are likely to be successful, which depends to a large extent on a dynamic community of users and contributing developers; hence one of the Incubator's objectives is to support creating a dynamic communities with "healthy" success indicators. However, for a casual visitor of an open source community it is hard to tell which project is likely to thrive and which are seriously at risk. There is a number of project management approaches for monitoring the risk of traditional projects. However traditional project monitoring focuses on human reporting that fits well a tightly coupled form of organization to ensure the quality of project reporting.

This kind of organization is hardly found in an *OSS* project, thus these approaches are only of limited use for a project leader or observer in an *OSS* project: most of the stakeholders are unfamiliar to each other and temporarily join the project with various constraints in time, place, and work synchronization. As a result the stakeholders collaborate by sending messages, data, and artifacts through a number of communication and development tools such as *SVN*, mailing lists and bug tracker; and respond to subscribed notifications similar to the publish subscribe schema in an event-based system.

The development processes in an Open Source Software project can be modeled as multi-agent event-based system: in this model the project stakeholders are agents, and their interactions and state changes are events. The event-based model and tool support allow to draw on process and artifact data from the global *OSS* project community that can help outsiders to better understand success and risk factors in the current state of a project and its community. This kind of data analysis can be especially helpful if human-based reports are suspected to be unsystematic, incomplete, or inconsistent.

In this paper we propose 1. a concept for modeling an *OSS* project as a multi-agent event-based system and 2. an initial measurement approach for event-based monitoring the rich collections of process events coming from the *OSS* project in order to better understand the actual benefit of tool-supported process

[1] http://news.netcraft.com/archives/web_server_survey.html, last access: 17/11/2006
[2] http://incubator.apache.org/

event data gathering, correlating and analyzing processes event data from the *OSS* community.

We report an empirical feasibility study investigating typical and easy-to-observe success and risk indicators of five *OSS* projects listed in the Apache Incubator. The results of this empirical study should motivate the discussion of current benefits and limitations of event-based software project monitoring and its application for balancing human-based reporting in commercial software project management.

2. Related Work

This section summarizes related work on event-based systems, different forms of organizing software development, and the team as focus of project monitoring.

2.1 Event-Based Systems

The *Distributed Event-based Systems*[3] (DEBS) community defines *event-based systems* (EBS) as:

Systems in which producers deliver events, and in which messaging middleware delivers events to consumers based upon their previously specified interest.

A prominent usage paradigm of *EBS* is the "publish and subscribe" paradigm, in which producers and consumers remain mutually anonymous. The consumers register for their interest in an event or a pattern of events, in order to be notified subsequently of any event, generated by a producer that matches their registered interest through the middleware [14].

Compared to the traditional "request/reply" paradigm, "publish and subscribe" imposes total decoupling [16]: (a) in space: participants' anonymity, (b) in time: participants are not required to be available at the same time, and (c) in synchronization: there is no interaction blocking a control flow. These advantages enable EBS to facilitate both scalability and system evolution [14, 17]. For these reasons, *EBS* are widely used for integrating loosely coupled application components, including sensors, device controllers, and databases. The use of *EBS* ranges from home security system to complex gas/oil pipeline remote monitoring systems.

A major trend in *EBS* adoption comes from business process monitoring, as dynamic business environments have been forcing many organizations to employ more sensitive system in order to be more responsive for capturing time-sensitive business opportunities such as in stock market monitoring system. A recent methodology to develop responsive systems is *SARI* (Sense and Respond Infrastructure) [11, 19], which facilitates processing internal and external events and using these events for triggering proactive actions as a response to changes in the business environment. SARI is controlled by Sense and Respond Loop as depicted in figure 1.

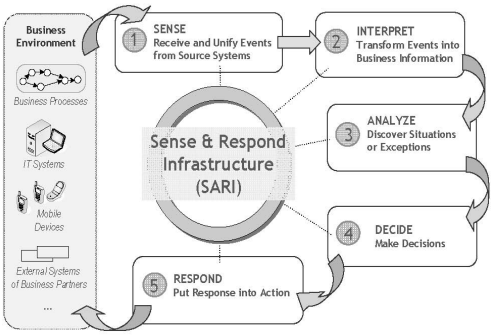

Figure 1: The "Sense and Respond" loops [22].

Challenges in engineering Event-based Systems

Mühl et. al. [9] and Fiege [14] report that *EBS* research and products are primarily focusing on scalability issues in terms of communication efficiency and system size, whereas basic problems of system engineering and management are often neglected.

Best practices for engineering *EBS* applications have not yet been agreed upon, most *EBS* applications were developed using reverse engineering emphasizing the need for more structured *EBS* development processes.

2.2 Open Source Project Characteristics

Open Source Software (OSS) projects have some typical characteristics which differ from closed source project as suggested in [1, 3, and 13]:

- Main contributions come from unpaid participants
- High level of participant distribution
- Weak formal: design, project planning and management
- Open code base and community-based project review and controlling

Although the work coordination in an *OSS* project may seem unorganized, there are several advantages suggested by the Open Source Summit[4] and Eric Raymond in his famous essay "The Cathedral and The Bazaar" [8] such as:

- Rapid development and massive peer review

[3] http://www.cs.queensu.ca/~dingel/debs05/

[4] http://linuxgazette.net/issue28/rossum.html

- Flexibility in using and modifying the source code for user interest
- Low-cost development and technology transfer
- Developer inheritance and the use of a reference implementation to help develop a standard.

A recent study [18] suggested that the social structure in *OSS* projects could provide some hierarchy of management and controlling based on *self-organizing patterns*. This makes *OSS* projects interesting objects for the empirical study of mechanisms software project management.

Several studies have used open source projects to better understand aspects of successful distributed development. Several studies observed the *OSS* projects by mining repositories such as mailing lists, *SVN/CVS*, bug databases [2, 5]. These studies clearly portrayed the development process and importance of community involvement as success factor in *OSS* projects.

2.3. The Team as Key Success and Risk Factor

In commercial contexts, a management executive may want to keep an overview over a portfolio of several projects and detect potential problems early. Thus they depend on the trustworthiness of monitoring critical success and risk factors in the project life cycle.

Similar to an *OSS* project, a commercial project also has to put a focus on the dynamics of the development team or project community as the prominent project success factor. The Standish group, in their famous Chaos report [20], discloses that over 44% of the respondents suggested the roles of the project participant (i.e. user involvement, executive management support, and competent committed staff) as the most critical success factor in a software project

Other empirical research reports [2] from distributed and collaborative software development environments (i.e., open source software projects) emphasize that project has also to consider issues of coordination, communication and other social structures. De Souza et al. [4] reported similar findings based on several large distributed *NASA* software projects.

According to the above mentioned research suggests that project participant reports are important sources of project information. However, Keil et al., [15] found a tendency among participants in troubled projects *not* to report problems objectively. This is a key risk issue, especially in large distributed and loosely coupled software projects.

3. A Comprehensive View of Software Project Monitoring

Project manager need software project monitoring to assess status of the software project in order to take necessary actions against certain risk conditions based on collecting selected software metrics along the project life cycle. To obtain actual status of the project, software project monitoring is required to supply accurate and comprehensive project information as the basis for analysis and decision making.

During development processes, the monitoring process should balance the observation from both (a) time-relevant process events data and (b) product-relevant artifact data as complement to each other. This combination will provide more accurate, less biased project information.

In general, there are two monitoring approaches: tool-based and human-based monitoring. Tool-based approaches are most suitable for monitoring frequently a large number of process event data, or when human resources for monitoring are hard to get. On the other hand, a human-based approach is best for a weekly/monthly process such as personal reporting.

However traditional software project management focuses on tracking formal achievements such as progress and financial obligation [10] and analyzing the merit of project participants based on routine personal reports and deliverables [10, 20].

As consequences most traditional project management is human-based monitoring, which often misses process and events information during project execution. This can be very risky, if a problem occurs, as Keil et.al [15] found participants to tend not to report the actual condition of the project. Thus additional data for comprehensive balanced reporting is needed before and during a crisis for raising issues well in advance to identify and mitigate project risks.

Current trends in distributed software development such as *OSS* project, signify three challenging conditions: 1. a large amount of process event data to be monitored, 2. shortage of human resources for monitoring, and 3. most important a loosely coupled project community as the result of global project work; consequently, monitoring such a system using a human-based approach only 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.

In this paper we propose the following research

issues:

- From project manager goals we derive a set of useful events and propose a way to measure, correlate and refine the collected data
- Investigate the potential contribution of project event data and their monitoring during the software development process?
- Propose an initial measurement model for event-based project monitoring and discuss the relevance of a sample of Open Source Project event data.

In this work we focus on process event data as object of monitoring. Events are processed into business information through event identification, event correlation and analysis. Later we perform measurement from event correlation metrics.

4. An Event-Based Project Monitoring Concept

The key risks condition which typically threatening OSS projects are the absence of key committers and the demotivation of project community. Many projects suffers heavy blow after abandoned by their key committers, this "brain drain" brings the project into troubles such as in Apache Xindice. Other risky situation also found in projects which have provided stable and useful releases in the past, but the target market may change and a specific technology may not particularly be interesting any longer (like native *XML* databases) which later demotivate the developers. These are obviously risky situations for the project and its users. However detecting such issues is a complicated task. Many different parameters have to be taken into consideration. This is particularly problematic if a large number of projects need to be monitored.

4. 1 Correlated Events as Status Indicators

An *OSS* project offers rich collection of process events and artifacts which could be observed during project lifetime. This collection may indicate the status of the project whether the project is in good condition or in deep trouble.

Based on our observations from many *OSS* projects, we found there are some community correlated events-that project leading teams routinely use as status indicator to assess an open source project, such as the following data:

- *Open issues, service delays:* Bugs and Issues are listed in the bug-tracking system, but the relevant/necessary fixes are not done in an appropriate time.
- *Proportions:* Calculate proportions of elements such as volume of mailing list postings, bugs per time slot, updates in the SVN, and use these metrics to compare projects to try to learn what a fine relationships are like.
- *Community activity and intensity:* indicate if a project has a dynamic community, e.g., the number of downloads compared to mailing list postings; the number of active power user (a user who help another user) in the mailing list and their email contribution intensity, developer interactions in (different) mailing lists.

A recent study [7] provided empirical support on the correlation of developers' (in particular, key committers') email contributions with project survivability. The study gathered the number of monthly developer mail contributions in the developer mailing list from two successful *OSS* projects (*HTTPD* and Tomcat) and two challenged projects (Xindice, and Slide) during the projects' life time which spanned a period of more than 6 years. The results indicate that both challenged projects were seems abandoned by their core developers/committers leading to brain drain in the project and demotivate other developers to remain in the project. As a consequence, the project became inactive.

These illustrating examples clearly describe that the correlation of events may provide a brief outlook of the project status of the project or signify early warning of risky situations.

4.3 Event-based OSS Project Monitoring

The development processes in an Open Source Software project can be modeled as multi-agent event-based system: in this model the project stakeholders are agents, and their interactions and state changes are events.

During the development processes, agents interact and move from one state to another triggered by events. They may act as producer who publish event through messaging middleware (i.e. mailing list, bug tracker, *SVN*), which then deliver events to other agents who act as consumers/subscriber based upon their previously specified interest.

Since most of the participants are: (a) unfamiliar with each other, (b) distributed around the globe with different time zone and work schedules, and (c) use various technologies and development-communication interfaces, as result, most of the messages and

deliverables during development processes are made with publish/subscribe-like interaction schemas as illustrated in Figure 2.

For example in a bug tracking process scenario, a user/developer who reports an issue can be considered as producer who send message about a bug existence into bug tracker (*bug_reporting_event*), then after performing some internal management operation, the bug tracker broadcasts the new bug information, e.g., through a mailing list (*new_bug_notification_event*). Later some subscribed user/developer may respond by making the diagnosis of the bug and send the result into the bug tracker (*bug_diagnoses_event*).

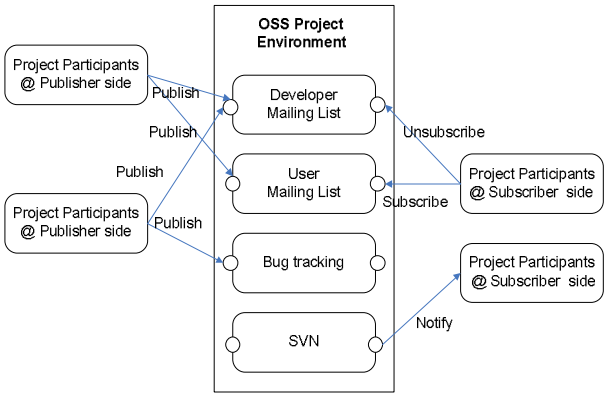

Figure 2: OSS project as event-based system

Event-based *OSS* project monitoring system focuses on exploiting process event data and their correlation during project execution to obtain relevant project information. The monitoring system starts with sensing the status change in project by identifying, and collecting events from *OSS* projects, later the system interprets collected events by correlating events and performs some measurements in order to transform the correlations into meaningful project information. Analysis is the next step based on the result of measurement which indicates the project status.

Afterwards the project status and the perception of the user of the system will define the decisions to be taken and what kind of responses should be executed. The decision stage is related to role specification, and decision guidelines. The response stage is basically as the result of the decision taken by the user which may have impact to the project community.

In this model we define two classes of the system user. These users are outsider who wants to have an outlook of the *OSS* projects for various purposes based on specific roles. The first user class are the common observer; they monitor *OSS* projects to enrich their knowledge and extract some valuable information for their own rationale about development processes, and decide whether a project is worth noting.

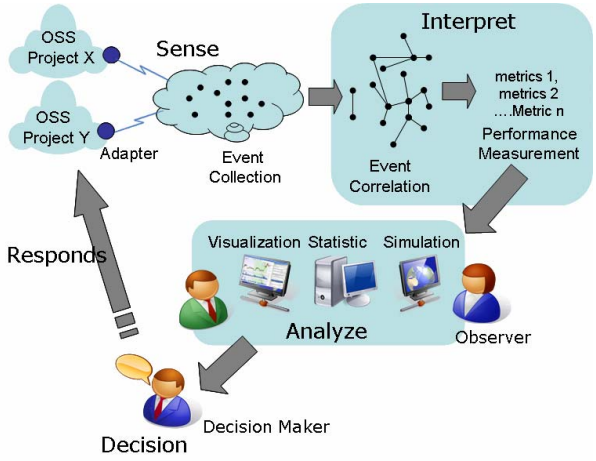

Figure 3: Event-based OSS project monitoring system

The second class of user are the decision maker; a decision maker is an observer with more significant roles, as they have stronger motivation for monitoring *OSS* projects. They are responsible in making decision and perform necessary actions as respond to status indicator of the projects. For example *IT* manager who wants to make feasibility study and risk assessment before using *OSS* solutions or to assure the survivability of currently use solutions. Other example of this user class is the project manager or project board such as in Apache Incubator who needs to monitor the status of many projects community consecutively as basis for risk mitigation, support allocation or dismantle unpromising projects in the incubator.

5. Measurement Process

Measurement is essential part of monitoring, in view of the fact that it provides meaningful project information for decision support. In this section we propose an initial structured model for measurement process using a simple scenario based on a project success factor and risk that commonly observed by expert in *OSS* community.

5.1 The Measurement Model

The stages in this model in general are motivated by (a) goal/question/metric (GQM) technique for defining

the monitoring goals and purpose; and (b) sense and respond loops for controlling the data collection and data analysis process, which consists of the following steps:

1. *Monitoring Definition.* The contextual level definition of monitoring is necessary to have clear direction of the measurement processes based.
2. *Data Collection.* Data collection is a process where the event-data are **sensed** from real system and **interpreted** for analysis purposes. This process consists of event identification, event gathering, and interprets collected event into project information. The interpretation consists of event correlation analysis, and performance measurement based on event correlation.
3. *Data Analysis.* **Analysis** is a process to extract the meaning of the combination of event during development process. The purpose of analysis is to discover current situation or exceptions of the project, and provide basis for taking appropriate **decisions** and **responses.**

5.2 Measurement Execution

As described in previous sections, one of the OSS project success indicators is developer contribution which points up the trust of the community for a project. The level of contributions can be observed by retrieving process event data in project repositories such as *SVN*, bug tracker, wiki and developer mailing lists.

In this work we focus on the developer contribution in the developer mailing list as an initial measurement example. We select developer mailing list due its importance level during development process such as described in [15] and the easiness of data retrieval.

5.2.1 Monitoring Definition

Goal: the purpose is to monitor the timeliness of developer contribution in developer mailing list from the view point of project leader.

Question: what is the current contribution intensity of developers in the mailing list?

Metric:

- Average number of emails per month. Monitoring this metrics will show observer the trend line of developer contributions. A positive trend line indicate the developers are active and show their willingness to the project
- Percentage of cases outside the lower limit of number of emails/month. This metric important as early warning of low level of developer contribution of developers in a month.

5.2.1 Data Collection

Identifying relevant event is the fundamental task in event-based monitoring. As events may come either from internal monitoring system or the observed systems, thus we should have better understanding of the observed system.

Luckham [6] proposed an *Event Processing Agent* (EPA) for identifying and interpret the event patterns. Start with identifying an *in_action* which could be an event that come and change the status of an agent and invoke a respond called *out_action*. We define incoming email from email producer into the processing email use case as *in_action* that should be monitored We called this event as *incoming_email* event with attributes as described in listing 1.

```
<incoming_email>
  <contributor>Jackson@hotmail.com</contributor>
  <date>2006-12-27</date>
  <header>session replication</header>
  <message> Hi there, I have …..</message>
</incoming_email>
```

Listing 1 Incoming_email event

The metrics of this monitoring specify some kind of early warning signal if the developer contribution is bellow a normal standard. Thus we define the second event which is an *out_action* called alert, as described in *XML* format in listing 2.

```
<alert>
  <type>Red</type>
  <date>2007-01-30</date>
  <header> Email contribution<60 </header>
</alert>
```

Listing 2 Alert event

The third step in *EPA* is to identify the correlation, behavior and rules of these events. Study of [7] suggests that two challenged projects (Xindice and Slide) went belly after consecutively acquired less then 70 email contributions/month. Thus we define the developer contribution status is considered as (a) "normal*"* if the number of email contribution is more or equal than 70 email or (b) "abnormal*"* if the number of email contribution is less then 70 which signifies low level of developer contributions.

The "abnormal" status indicates an early warning of a risky situation. Based on this rule we define following algorithm to depict the behavior of the

monitoring.

At the end of each month the system will triggers alert which either green, yellow or red based on number of incoming email counted from the developer mailing list, which signify the developer contribution status.

```
foreach(month in year)
  {
  for(day=beginOfMonth[month];
  day<=endOfMonth[month];day++)
   {
    if (day==beginOfMonth)
     {counter[month,year]=0}
     else
    {if(incomingEmail&&
     emailHeader!="announcement"))
      {counter[month,year]=++;}
     }
    }
    if(counter[month,year]>=70){alert(green);}
    elseif(counter[month,year]>=60)
      {alert(yellow);}
    else {alert(red);}
  }
```

Listing 3 Behavior rules

5.2.1 Data Analysis and Discussion

We collected the *incoming_email* events on monthly basis from five *OSS* projects in *Apache Incubator*, which are *Woden*[5], *OpenJPA*[6], *Lucene.Net*[7], *Roller*[8], and *Ode*[9]. We retrieved data from developer mailing lists beginning in March 2006 until December 2006 for each project.

The result as depicted in figure 4 reveals that Wooden can be considered as a healthy as there is no alert triggered with average 224.3 email contributions per month and standard deviation = 112. Open JPA, Roller and Ode have total average of 20% of cases of outside lower limit and total average of 142.2 emails per month, which seems to be normal as these projects are in incubation and may attract more developer contributions in the future.

However Lucene.Net seems to be in trouble, since in the last 8 months of development there are 6 "red" alerts (depicted as red dots in figure 4) or 75 % of case of outside lower limit which indicate the low contribution of developer into the mailing list with average of 40.6 emails/month which is more then 120 email bellow of total average email contributions of the other 4 projects.

We investigate further the situation faced by Lucene.Net, by locating the alarm triggered events in which periods, and observe other time-relevant correlated events in developer activity such as product releases. We found that in the last 8 months, Lucene.Net[10] has produced several important events, such as 1 major release (v. 2.0), 1 minor release (v. 1.9.), and two micro releases (v 1.9.0 and v 1.9.1).

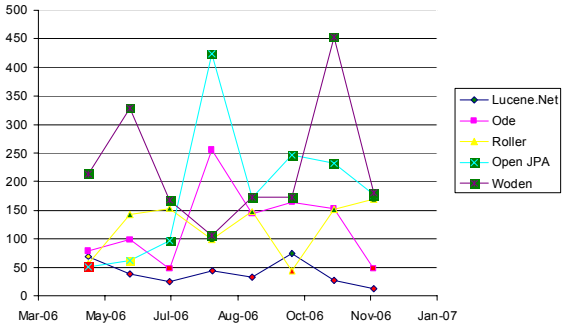

Figure 4 Developer emails contribution

We found that there is a contradiction in challenged project (Lucene.Net) which in generally well developed and provide regular releases, and appreciated by the user community, but it might actually be driven by very few active committers since the developer contribution in the mailing list is very low compare to other observed projects. This risky situation of Lucene.Net in the long run could threaten the project survivability, which required more attention from the Apache Incubator.

7. Conclusion and Future Work

Event-based OSS software project monitoring is an approach of using EBS in the context of systems engineering and software project management. This paper provides an initial concept for project monitoring based on process event data.

The result of our initial feasibility study using developer contributions into the developer mailing list reveals evidence that tool-supported event based monitoring can act as the supplement of traditional project monitoring. In principle this concept can be implemented in commercial software project context, which share the same characteristic with OSS project

[5] http://incubator.apache.org/woden/
[6] http://incubator.apache.org/openjpa/
[7] http://incubator.apache.org/lucene.net/
[8] http://incubator.apache.org/roller/
[9] http://incubator.apache.org/ode/

[10] Last retrieved at 19/01/2007, from http://incubator.apache.org/lucene.net/

such as in global or distributed software project development.

However this concept poses limitations such as the lack of empirical proven events correlation as indicator of the project status, which are many of this correlation should be manually investigated from various OSS communities. As a tool-based monitoring approach, event based project monitoring system can provide early or complement indicator of project status, but the part of analysis, decision and response should be accompanied or emphasized by human wisdom for better result.

In future work, the concept should be enriched by more event correlation as project status indicator and further empirically investigated by implementation into some real life scenarios and some extension to commercial software project domain.

Acknowledgement

This paper is part of larger research called *Monitoring Distributed Software System Development and Operation*. More details of the event based OSS project monitoring concept can be seen in our technical report[11].

This work has been partly supported by Technology-Grant-South-East Asia No. 124-2/BAMO/2005, financed by ASIA-Uninet in co-operation with the Austrian Council for Research and Technology.

References

[1] A. Capiluppi, P. Lago, and M. Morisio. *Characteristics of Open Source Projects*. In proceeding of the 7th European Conf. Software Maintenance and Reengineering (*CSMR 03*), IEEE CS Press, 2003, 317–330.

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

[3] C. Gacek, B. Arief. *The many meanings of open source*, *IEEE Software*, 21, 2004, 34-40.

[4] C.R.B de Souza, D. Redmiles, G. Mark, J. Penix, M. Sierhuis. Management of Interdependencies in Collaborative Software Development. In proceeding of the International Symposium on Empirical Software Engineering (ISESE'03), 2003.

[5] D. German & A. Mockus. *Automating the Measurement of Open Source Projects*. in Proceedings of the 3rd Workshop on OSS Engineering, Portland, May 2003

[6] D. Luckham, The Power of Event: An Introduction to Complex Event Processing in Distributed Enterprise System, Addison Wesley, 2002

[7] D. Wahyudin, A. Schatten, K. Mustofa, S. Biffl, A. Tjoa, *Introducing Health Perspective In Open Source Web-Engineering Software Projects, Based On Project Data Analysis*. The 8th International Conference on Information Integration and Web-based Applications & Services (IIWAS2006), Yogyakarta, Indonesia. 2006.

[8] E. Raymond, *The Cathedral and the Bazaar*, O'Reilly, 1999.

[9] G. Mühl,, L. Fiege, P. Pietzuch, *Distributed Event-Based Systems*, Springer, 2006

[10] J. Philips, *IT Project Management: On Track from Start to Finish*, Mc Graw Hill, 2002

[11] J. Schiefer , A. Seufert, *Management and Controlling of Time-Sensitive Business Processes with Sense & Respond*, In proceeding of the International Conference on Computational Intelligence for Modelling, Control and Automation, 2005

[12] J. Schiefer, C. Mc Gregor, Correlating Events for Monitoring Business Processes, International Conference on Enterprise Information Systems, Porto, 2004.

[13] K.Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly, 2005.

[14] L. Fiege, *Visibility in Event-based System*, PhD Dissertation, Darmastadt University of Technology, 2005

[15] M Keil, H.J. Smith, S.Pawlowski, L. Jin. *"Why Didn't Somebody Tell Me?" Climate, Information Asymmetry, and Bad News about Troubled Projects*. ACM SIGMIS Database,35, 2 (spring, 2004), 65-84.

[16] P. Eugster, P. Felber, R. Guerraoui, A. Kermarrec *The Many Faces of Publish/Subscribe*. ACM Computing Surveys, 2003. 35(2).

[17] P. Pietzuch, *HERMES. A Scalable Event-based Middleware*, PhD Dissertation, Queens' College University of Cambridge, 2004

[18] S. Valverde, G. Theraulaz, J. Gautrais, V. Fourcassie, R.V. Sole. *Self-Organization Patterns in Wasp and Open Source Communities*. IEEE Intelligent System, 21, 2, (March 2006).

[19] T.M. Nguyen, J. Schiefer, A..M. Tjoa, *Sense & response service architecture (SARESA): an approach towards a real-time business intelligence solution and its use for a fraud detection application*. In Proceedings of the 8th ACM international workshop on Data warehousing and OLAP, 2005.

[20] W. Royce. *Software Project Management: A Unified Framework*. Pearson Education, 2000.

[11] http://www.ifs.tuwien.ac.at/~dindin