Empirical Project Monitor: A Tool for Mining Multiple Project Data

Masao Ohira[†], Reishi Yokomori[‡], Makoto Sakai[††],
Ken-ichi Matsumoto[†], Katsuro Inoue[‡], Koji Torii[†]
[†] Nara Institute of Science and Technology
ohira@empirical.jp, {matumoto, torii}@is.aist-nara.ac.jp
[‡] Graduate School of Information Science and Technology, Osaka University
{yokomori, inoue}@ist.osaka-u.ac.jp
[††] SRA Key Technology Laboratory, Inc.
sakai@sra.co.jp

Abstract

Project management for effective software process improvement must be achieved based on quantitative data. However, because data collection for measurement requires high costs and collaboration with developers, it is difficult to collect coherent, quantitative data continuously and to utilize the data for practicing software process improvement. In this paper, we describe Empirical Project Monitor (EPM) which automatically collects and measures data from three kinds of repositories in widely used software development support systems such as configuration management systems, mailing list managers and issue tracking systems. Providing integrated measurement results graphically, EPM helps developers/managers keep projects under control in real time.

1 Introduction

In software development in recent years, improvement of software process is increasingly gaining attention. Its practice in software organizations consists of repeatedly measuring the development activities, finding potential problems in the processes, assessing improvement plans, and providing feedback into the processes. Project management for effective software process improvement must be achieved based on quantitative data.

Many software measurement methods have been proposed to better understand, monitor, control, and predict software processes and products [4]. For instance, the Goal-Question-Metric (GQM) paradigm [2] provides a sophisticated measurement technique. GQM guides to set up measurement goals, create questions based on the goals, and determine measurement models and procedures based on the questions. The measurement based on GQM is a logical and reasonable method.

However, in its practice, members who participate in measurement activities need to strive for the measurement processes on every last detail. Data collection for measurement in general requires high costs and collaboration with developers. It is difficult to collect coherent, quantitative data continuously and moreover to utilize the collected data for practicing software process improvement. Few studies have proposed measurement tools for dealing with a number of project data especially in terms of a large-scale software organization.

As a measurement-based approach to the above issues, we have been studying empirical software engineering [1, 3] which evaluates various technologies and tools based on quantitative data obtained through actual use. Our goal is to develop an environment composed of a variety of tools for supporting measurement based software process improvement, which we call Empirical software Engineering Environment (ESEE).

In this paper, we introduce Empirical Project Monitor (EPM) as a partial implementation of ESEE, which automatically collects and measures quantitative data from three kinds of repositories in widely used software development support systems such as configuration management systems, mailing list managers and issue tracking systems. Collecting such the data in software development automatically and providing integrated measurement results graphically, EPM helps developers/managers keep their projects under control in real time.

2 Empirical Project Monitor (EPM)

We have developed Empirical Project Monitor (EPM) [9] which automatically collects and analyzes data from multi-

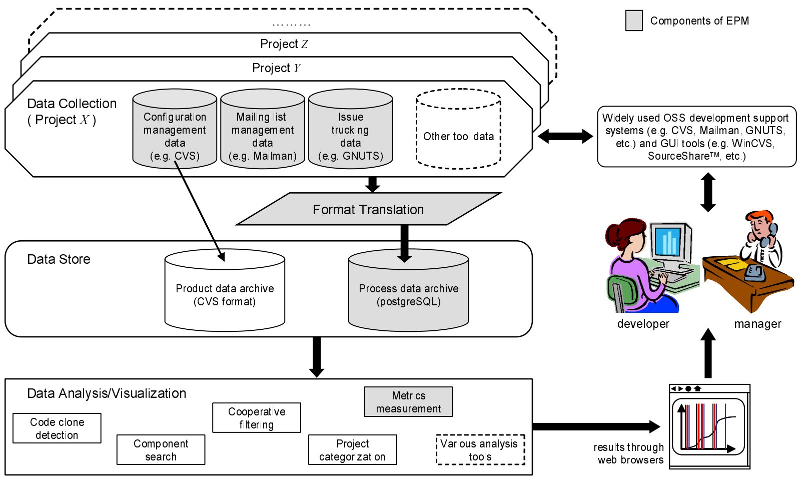
Components of EPM
....
Project Z
Project Y
Data Collection (Project X)
Configuration management data (e.g. CVS)
Mailing list management data (e.g. Mailman)
Issue trucking data (e.g. GNUTS)
Other tool data
Widely used OSS development support systems (e.g. CVS, Mailman, GNUTS, etc.) and GUI tools (e.g. WinCVS, SourceShare™, etc.)
Format Translation
Data Store
Product data archive (CVS format)
Process data archive (postgreSQL)
developer
manager
Data Analysis/Visualization
Code clone detection
Component search
Cooperative filtering
Project categorization
Metrics measurement
Various analysis tools
results through web browsers

Figure 1. The architecture of EPM in the ESEE framework

ple software repositories. Figure 1 shows the architecture of EPM in the ESEE framework. The ESEE framework is designed for supporting measurement based process improvement in software organizations by providing various pluggable tools. EPM consists of four components according to the ESEE framework: data collection, format translation, data store, and data analysis/visualization. This section describes an overview of EPM and the basic data flow through EPM.

Automatic data collection: EPM automatically collects multiple project data from three kinds of repositories in widely used software development support systems. For instance, EPM collects versioning histories from configuration management systems (e.g. CVS[1]), mail archives from mailing list managers (e.g. Mailman[2], Majordomo[3], fml[4]), and issue tracking records from (bug) issue tracking systems (e.g. GNATS[5], Bugzilla[6]). Because these data are accumulated through everyday development activities using common GUI tools (e.g. SourceShare^{TM}[7], WinCVS[8]), developers/managers do not need additional work for data collection. Also, it dose not take high costs to introduce EPM into projects/organizations because the systems as the sources of data collection are open source freeware.

Format translation and data store: EPM converts the collected data into the XML format called the standardized empirical software engineering data, so that EPM can deal with not only the above three kinds of software repositories but also various kinds of repositories according to purposes for measurement. Data from other systems are available by small adjustments of parameters. The data converted into the XML format is stored in the PostgreSQL[9] database.

Analysis and visualization: EPM analyzes the data stored in the PostgreSQL database. For instance, in order to analyze data related to CVS, EPM extracts the process data about events such as checkin/checkout, transitions of source code size, version histories of components, and so forth. Then, EPM visualizes various measurement results such as the growth of lines of code and the relationship between checkin and checkout. EPM also provides summaries of each repository such as information of CVS logs. All the measurement results are available through using common web browsers (e.g. see Figure 2), so that users are easy to share the results.

In this way, EPM supports users to obtain quantitative data at low cost in real time and provides them with various measurement results for understanding the current development status. This would help users keep their projects under control.

3 Visualizations of measurement results

Data mining techniques for software repositories have been proposed to understand reasons of software changes [7], to identify how communication delay among developers in physically distributed environments have effects on software development [8], to detect potential software changes and incomplete changes [11], and so forth. In contrast to these tools, the features of EPM are to visualize

[1]CVS, http://www.cvshome.org/
[2]Mailman, http://www.list.org/
[3]Majordomo, http://www.greatcircle.com/majordomo/
[4]fml, http://www.fml.org/index.html.en
[5]GNATS, http://www.gnu.org/software/gnats/
[6]Bugzilla, http://www.bugzilla.org/
[7]SourceShare^{TM}, http://www.zeesource.net/
[8]WinCVS, http://wincvs.org/
[9]PostgreSQL, http://www.postgresql.org/

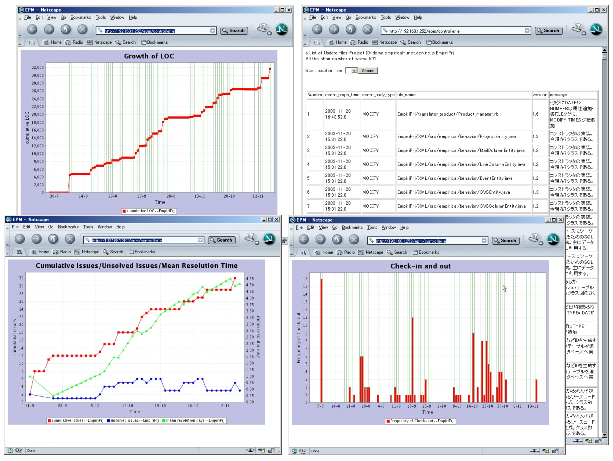

Figure 2. Measurement results through web browsers

combinations of measurement results from three kinds of software repositories and to be able to deal with data from multiple projects simultaneously.

3.1 Combinations of measurement results

In addition to providing visualizations of measurement results from each software repository, EPM also visualizes combinations of measurement results from three kinds of repositories. The followings show two examples of them.

Bug issues and checkins: Figure 3 represents the relationship between the transition of the cumulative total of issues (the line graph) and the time of checkins (the grayed vertical lines on the X-axis) in our EASE project [6]. The number of issues and checkins are measured from the repository in GNATS and CVS respectively. A checkin often occurs after bug issues are reported because developers try to modify or resolve the issues. The graph helps users (developers/managers) remember the situation where issues by every file versions were raised. To the contrary, the file itself which is checked in CVS may include some bugs if the graph indicates that there are issues after checkins.

Bug issues and e-mails among developers: Figure 4 illustrates the communication history among developers in the EASE project. The black line graph is the transition of the cumulative total of e-mails exchanged through using Mailman. The vertical shorter/longer dashed lines represents when bug issues were raised/resolved. The light-gray vertical lines mean when the checked-in files by developers were uploaded to CVS. From the graph, users can confirm the state of the communication among developers and identify the file versions which might have problems. Because discussions on issues become active usually when issues are reported to an issue tracking system, the communication itself among developers might have problems if many issues are reported but developers did not discuss on the issues. Communication problems among developers bring the decrease of software productivity and reliability [8].

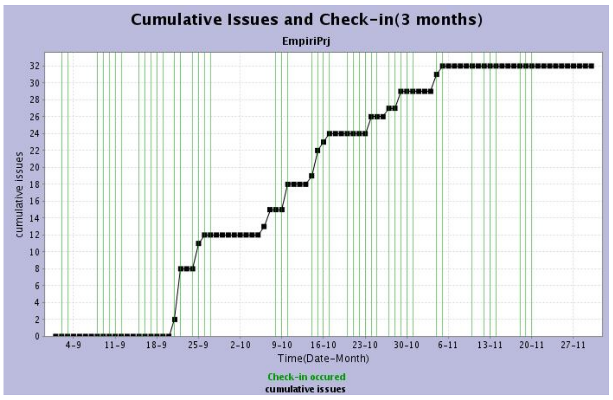

Figure 3. Relationship between issues and checkins

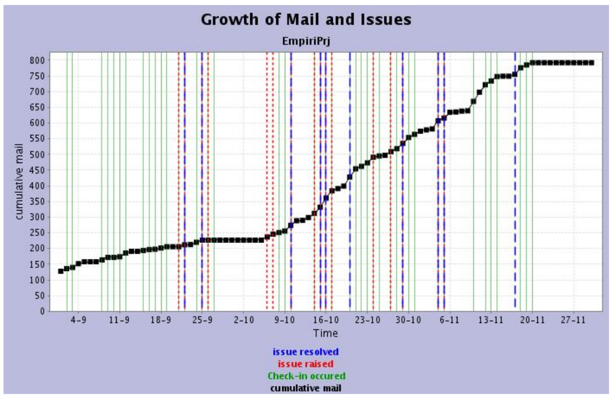

Figure 4. History of bug issues and e-mails among developers

The integrated measurement results based on data from configuration management systems, mailing list managers, and issue tracking systems help developers understand current and past events in development activities.

3.2 Visualizations of multiple project data

EPM has the capability to visualize multiple project data. Comparing current projects with past ones would be helpful for managers to estimate the progress of projects and to detect the unusual status in projects.

Comparison of measurement results among multiple projects: EPM makes measurement results comparable with multiple projects. Figure 5 represents the relationship of the growth of lines of code between two project (the upper line: SPARS [10], the lower line: EASE). The both projects have been proceeding under the collaborative research with authors' universities and some software companies. Some researchers and developers have been participating in the both projects. Actually although the both have different purposes and aspects, suppose here that they have been developing software systems respectively under similar conditions. The project managers can confirm some common characteristics and roughly estimate the progress of the later project (EASE) from the graph. For instance, SPARS has the two phases in which it have evolved rapidly for releasing major versions. EASE has just released the first major version. The managers are easy to guess the near future of the progress of EASE: the development of EPM will stop for a while to test the EPM, to reconsider the design, and so forth.

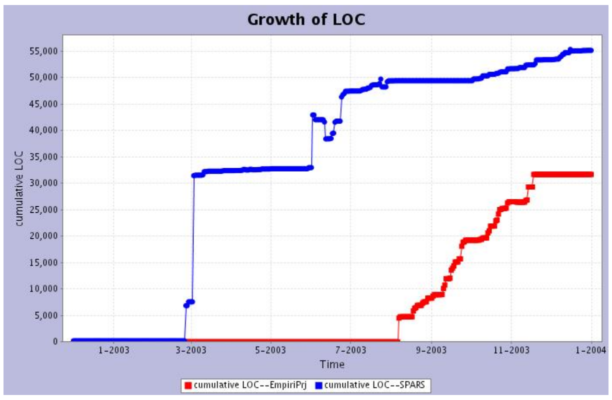

Figure 5. Comparison of two projects

Distribution maps of multiple projects: Using measurement results from three kinds of repositories in multiple projects, EPM can generate distribution maps. Figure 6 is a distribution map using 100 Open Source Software Development (OSSD) projects data collected from SourceForge.net[10,11], which represents the relationship between lines of code (the X-axis) and number of checkins (the Y-axis). Suppose here that these projects are managed by one software organization. The graph can be used for helping managers identify "unusual" projects which indicate extreme high or low values.

[10]SourceForge.net, http://sourceforge.net/

[11]We selected the 100 projects in Figure 6 randomly from the list of most active projects in SourceFroge.net.

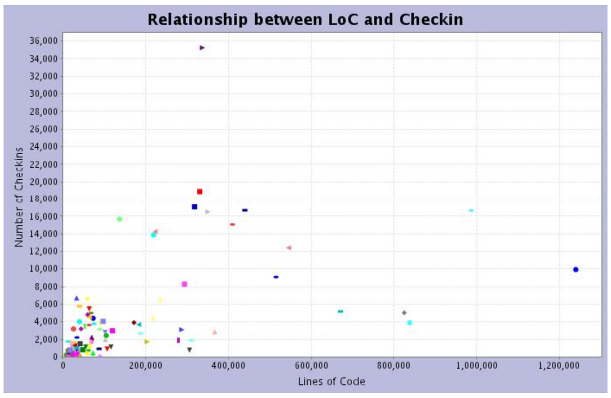

Figure 6. Distribution map of 100 OSSD projects

3.3 Customizations of measurement parameters

EPM currently provides users with the five types of graphs including Figure 3-5 and two types of summary information from CVS and mailing list data. Users have the choice of visualizing single project data or multiple project data according to purposes of analysis. EPM also provides an interface to customize queries for the PostgreSQL database. Using the database schema for EPM which is open to the public, users are able to input SQL sequences and to create bar graphs, line graphs, and distribution maps such as Figure 6. Because we would like to support various projects and organizations which have own problems respectively, we decided to provide the minimum types of graphs and summary information rather than to provide a lot of them in advance. After feedback from software organizations using EPM, we will add other types of graphs in the near future. Currently EPM can be viewed as a tool for exploratory data analysis [5].

4 Discussion

In this section, we report a case study of applying EPM to our project itself, in order to observe the actual usage of the pre-defined 5 types of graphs mentioned above. We have interviewed four developers on the advantages and the disadvantages of using EPM. The development environment of this project is summarized in Table 1.

One of the advantages is that the graphs make developers easy to understand the status of the project by identifying distinctive parts indicated in the graphs. For instance, the part of the flat line in the LoC graph reminded them why the development seemed to be stopped. In fact, all developers were on a business trip at the time. This could help them

Table 1. EPM development project

Target project	EPM development project
Programming language	Ruby, Java
Number of developers	4
Repositories	CVS, Mailman, GNATS
Development period	3 months
Preparation period	1 week

increase the accountability for their managers. Other one is that the graphs generated in real time motivated developers to fix bugs, since they could be aware that there were still unresolved issues.

In contrast to these advantages, some problems related to the usage of EPM have been found. One is that visualizations are too complicated to understand the status of the project in some cases. For instance, developers could not distinguish which file versions corresponded to which vertical lines in Figure 3, since one developer checked in CVS for backup of his files every day and therefore a number of checkins occurred. In this case, developers might need to use two CVS (e.g. one is for software release and another is for backup).

The above results are still the initial evaluations for EPM. EPM will be introduced in some software companies in the near future. We intend to evaluate the usefulness of EPM with respect to (1) the effects on software development and process improvement by providing measurement results from multiple software repositories, and (2) the benefit of giving the capability to manage multiple projects.

5 Conclusion and Future Work

The goal of this research is to construct an environment for supporting measurement based software development according to the ESEE framework. In this paper, we introduced Empirical Project Monitor (EPM) as a partial implementation of ESEE, which helps developers/managers keep projects under control by providing various visualizations of measurement results related to project activities. Nowadays, we can gather and analyze massive data on software development in a large scale using rapidly growing hardware capabilities. By analyzing such the huge data collected from thousands of software development projects, we would like to provide useful knowledge and benefit not only to individual developers/managers but also to organizations.

Empirical study on software development is an active area in the field of Empirical Software Engineering (ESE). But the approaches of ESE have not been sufficiently applied to software development in software industry although companies hold many problems. The data related to software development from the industrial world has seldom been provided with university's research. We are collaborating with some software development companies as the EASE project. Therefore, it would be a strong trigger for going beyond the obstacle of the technical progress in software engineering.

Acknowledgment

This work is supported by the Comprehensive Development of e-Society Foundation Software program of the Ministry of Education, Culture, Sports, Science and Technology. We thank Satoru Iwamura, Eiji Ono and Taira Shinkai for supporting the development of Empirical Project Monitor.

References

[1] A. Aurum, R. Jeffery, C. Wohlin, and M. Handzic. *Managing Software Engineering Knowledge*. Springer, Germany, 2003.

[2] V. Basili. *Goal Question Metric Paradigm, in Encyclopedia of Software Engineering (J. Marciniak ed.)*, pages 528–532. John Weily and Sons, 1994.

[3] V. Basili. The experimental software engineering group: A perspective. ICSE'00 award presentation, June 2000. Limerick, Ireland.

[4] L. Briand, C. Differding, and D. Rombach. Practical guidelines for measurement-based process improvement. Technical Report ISERN-96-05, Department of Computer Science, University of Kaiserslautern, Germany, 1996.

[5] S. Card, J. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*. Morgan-Kaufmann Publishers, San Meteo, CA, 1999.

[6] EASE. The EASE (Empirical Approach to Software Engineering) project, http://www.empirical.jp/intex-e.html.

[7] D. German and A. Mockus. Automating the measurement of open source projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering*, pages 63–67, Portland, Oregon, 2003.

[8] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter. An empirical study of global software development: Distance and speed. In *Proceedings of the 23rd international conference on Software engineering (ICSE'01)*, pages 81–90, Toronto, Canada, 2001.

[9] M. Ohira, R. Yokomori, M. Sakai, K. Matsumoto, K. Inoue, and K. Torii. Empirical project monitor: Automatic data collection and analysis toward software process improvement. In *Proceedings of 1st Workshop on Dependable Software System*, pages 141–150, Tokyo, Japan, 2004.

[10] SPARS. The SPARS (Software Product Archiving and Retrieving System) project, http://iip-lab.ics.es.osaka-u.ac.jp/SPARS/index.html.en.

[11] T. Zimmermann, P. Weissgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering (ICSE'04)*, Edinburgh, Scotland, UK, 2004 (to appear).