Fine-grained Analysis of Global Software Development Process

Shuji Morisaki, Tomoko Matsumura, Kimiharu Ohkura, Kyohei Fushida,
Shinji Kawaguchi, Hajimu Iida

Graduate School of Information Science, Nara Institute of Science and Technology,
8916-5 Takayama, Ikoma, Nara, Japan

Abstract

This paper proposes a method for micro process analysis of ongoing distributed software development. Micro process analysis firstly identifies process instances by matching logs of software development tools with pre-defined micro (fine-grained) process models. Micro process metrics can be measured from identified process instances. In distributed or multi-sited software development environment, micro process metrics provide project manager with more visibility to precise process execution and also to precise monitoring. As an empirical study, we retrospectively applied micro process analysis to bug fixing process in organizations attending a multi-site project of commercial software development. In the study, process instances are extracted from histories of CVS (a source code configuration management system) and GNATS (a bug tracking system) based on a bug fixing procedure predefined in the project. We confirmed that extracting and visualizing process instances helps to identify accuracy of process executions as well as accuracy of logging.

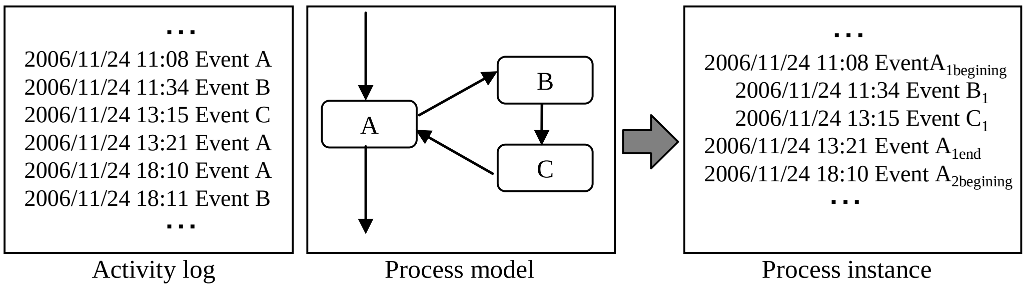
1. Introduction

Multi-site software development becomes more and more popular. In multi-site software development, low visibility to the development process easily lead to low software quality or delivery slippage. Thus, methods and tools to visualize product status and change histories in distant site have been proposed. Froehlich et al. proposed Augur [2] in order to provide more visibility and insight by visualizing source code produced date for providing visibility in distributed software development team. Zimmerman et al. proposes ROSE[6] that visualizes source code version history. While such practices are broadly made to product itself, procedural aspects of the product manipulations as fine-grained software process were not sufficiently studied in realistic situations. One major reason is that collecting process data automatically, including efforts and durations of activities, is difficult and process data are mainly identified and manually collected.

Meanwhile, studies of evaluating or assessing software process executions fall into two categories. One is the qualitative assessment approach like CMMI [1] which requires highly trained assessors. The other is quantitative evaluation of process execution histories based on artifacts such as change management log or daily activity reports. However, granularity and resolution of the target process are limited by those of the documents as the data source.

In the article [4], we are proposing retrospective (i.e., after development project has finished) analysis of the fine-grained process as *micro process analysis* We refer to a set of execution histories, generated by software development tools such as configuration management systems and integrated development environments, as fine-grained activity log. Applying grammatical process model to the activity log leads to formal and quantitative analysis of software process.

This paper proposes applying micro process analysis to ongoing distributed, or multi-sited, software development in order to provide visibility and opportunities to ask accurate process executions and monitoring. We applied our methods to a commercial multi-sited project data and extracted *process instances* (process fragments identified in an activity log) By visualizing the process instances, we found that there were one issue of logging and two issues of process executions in the project. The activity logs in the study are collected from CVS (a configuration management system) and GNATS (a bug tracking system). The process model in the study is based on the bug fix procedure determined in the project. We used a prototype tool for extraction and visualization of process instances.



...
2006/11/24 11:08 Event A
2006/11/24 11:34 Event B
2006/11/24 13:15 Event C
2006/11/24 13:21 Event A
2006/11/24 18:10 Event A
2006/11/24 18:11 Event B
...

...
2006/11/24 11:08 Event$A_{1begining}$
2006/11/24 11:34 Event B_1
2006/11/24 13:15 Event C_1
2006/11/24 13:21 Event A_{1end}
2006/11/24 18:10 Event $A_{2begining}$
...

Fig. 1 Process instance extraction from process model and activity log

2. Micro process analysis

2.1 Activity log and Software Process Model

In this subsection, the definition of the activity log is given. Activity log *E* is defined as a set of sequential events. Each event *e* is observed and collected through execution of software development. An event *e* ($\in E$) consists of 3-tuple <*t*, *s*, *a*> where *t* represents a time stamp of the event *e* occurred, *s* represents a type name of the event *e*, and *a* (=$\{a_1, a_2, \ldots a_n\}$) represents a set of attributes of event *e*. Members in *a* depends on the type of *e*. We don't set strict assumptions to type and granularity of events. An event is just assumed to be collected and recorded automatically by software development tools, so that every event has a type associated to its data source, such as change management log of intermediate and delivered product. Below is an example of a notation in an activity log of a document change management tool.

```
<2006/12/2  13:07,  update,  {file  name
= "detailed_design_function_document.doc",
submitted by ="Smith", comment = "fixes on
the item 0004 in 2nd inspection">
```

Event set *E* can be obtained directly from development tools logs. Project monitoring environment such as EPM [5] and Hackystat[3] enable automatic collection of such activity logs. EPM provides histories of CVS, GNATS and Mailman by default without specific modification to each tool, while Hackystat provides more detailed histories including keystroke logging by adding *sensors* to each tool.

We represent a software process as a *grammar* of activity sequences. Process model in Fig.1 shows an example of software process model described as an activity diagram. The model describes that activity A involves sequential sub-activities B and C.

2.3 Extracting Process Instances

A process instance is an actual sequence of activities that happened in the project. Process instances are extracted by parsing activity log based on a specified process model. Events corresponding to begin-end pair or whole execution of activities should be collected in activity log. Fig. 1 shows an example of process instance extracted according to the process model in Fig. 1. In this case, each event in the activity log corresponds to activity begin/end of *A*, or whole execution of *B* or *C*. We see that event at 11:08 as *A-begin* and event at 13:21 is an *A-end*. The events for *B* and *C* are observed between *A-begin* and *A-end*.

Micro process metrics, such as number of activity omissions or incorrect sequence, can be obtained from identified instances. They are useful for evaluating quality of the process as well as the quality of the logging itself.

Though there are limitations in interpreting and extracting process instances due to ambiguity and uncertainty that naturally exist in software processes, micro process metrics enables to assess quality of software process automatically and intuitively.

2.4 Analyzing distributed on-going projects

Micro process analysis can be applied to project in progress as well as retrospective project. Especially in distributed software development, micro process analysis provides valuable opportunities to improve remote process and also to improve quality of activity log at low cost.

Fig. 2 shows an overview of distributed software development using micro process analysis. Fig. 3 shows activity diagram of the procedures of on-time analysis. First, project manager and development sites agree to follow the prescriptive process models. Agreed process models are distributed to each site. Second, each site collects activity log and sends them to the project manager periodically. Process instances are extracted from those activity logs. If project manager recognizes degrade of process quality such as omissions of activity, or incorrect sequence, s/he can ask the site members to refer the specified process model, or s/he may reconsider the process model itself. Project manager can also recognize degrade of collected activity log, e.g. many logs are recorded at one time (not on-time), or many mismatch of begin/end of the activities are found. In such cases, manager

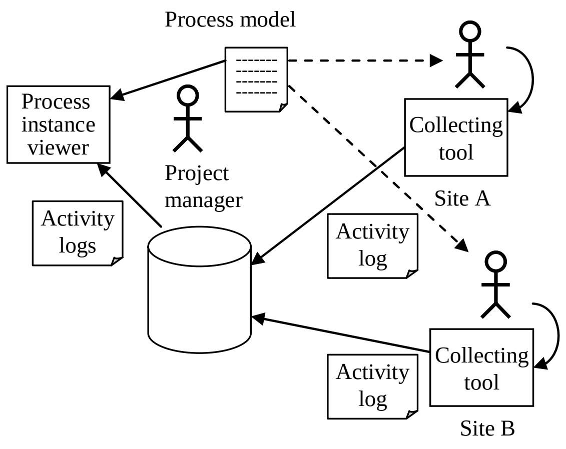


Fig. 2 Overview of real time process analysis

should check tool settings or operational prescription for activity logging in the target site.

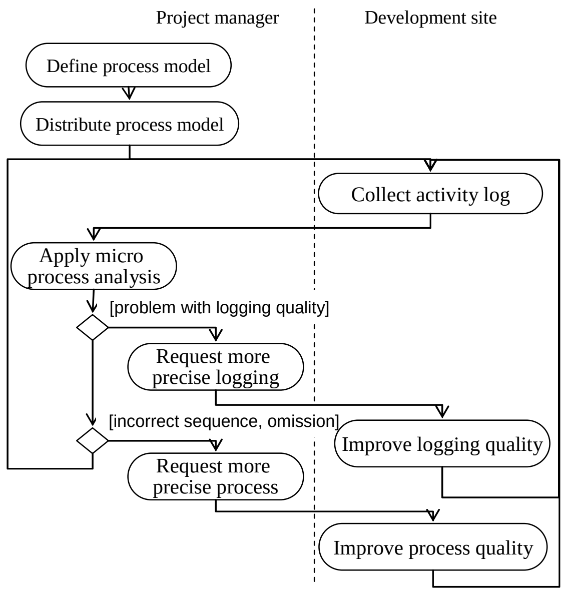
3 Example Study

3.1 Activity log and process model

As an example study, we applied the micro process analysis to a commercial software development project that follows traditional water fall model. The software was developed by five companies in six distributed development sites (One company had two sites and the others had one site). We have analyzed the activity logs collected from one of those sites. Used process model was preliminary defined and committed by those companies.

The activity logs consist of update histories of CVS (source code configuration management tool) and state-change histories of GNATS (bug tracking tool) that were integratedly collected by EPM[5]. Duration of the target process is approximately one month including later steps of coding phase followed by unit testing and integration testing phases. The process model in Fig. 4 is described based on the bug fixing procedures to which the members were requested to follow during the debugging process. The process consists of code fixing activities corresponding to operation to CVS and registration and closing of a bug entry corresponding to operations to GNATS.

Table 1 shows attributes to be registered by developers in each activity in Fig. 4. When a developer finds a bug, s/he registers statements of bug (“register to GNATS” in Fig. 4). A statement includes detected date time, assigned developer, description of bug and priority. A unique identifier is automatically assigned to the bug and status is set to be “get started” by GNATS. The assigned developer checks out source code from CVS to fix the problem if necessary (“check



Project manager
Development site
Define process model
Distribute process model
Collect activity log
Apply micro process analysis
[problem with logging quality]
Request more precise logging
[incorrect sequence, omission]
Improve logging quality
Request more precise process
Improve process quality

Fig. 3 Activity diagram of multi-site process analysis

out source code” in Fig. 4). The developer locates and corrects the defect then confirms the fix by testing. If the developer decides that the bug is fixed, s/he checks in the modified source code to CVS (“check in source code” in Fig. 4). The bug identifier assigned at “Register to GNATS” activity is manually specified as a comment of checking in. The developer changes the bug status in GNATS to “resolved” (activity *Change status in GNATS to “resolved”* in Fig. 4).

3.2 Visualization of process instances

Fig. 5 is visualized output of process instance extraction tool. In Fig. 5, vertical axis and horizontal axis represents date time and bug identifier respectively. The process instances are plotted according to bug ID and date of the start time. Events in each process instance are plotted according to time stamp obtained from activity log. Diamonds, squares, triangles and Xs in Fig. 5 represent “Remember detected date time”, “Register to GNATS”, “Check in source code” and “Change status in GNATS to resolved” respectively. Each *valid* (no omission or miss-ordering) process instance is identified as a sequence of these four marks.

Both of quality of process executions and quality of logging were observed in extracted instances. Fig. 5(a) is an example of low quality of logging. Process data for bugs no. 96 through no. 108 were recorded at once though each bug is actually detected, fixed and closed at different date time. Fig. 5(b) and (c) are examples of low quality of process executions. The process instance surrounded by dotted box (b) in Fig. 5 is an

Table 1 Attributes of processes (in part)

Activity	Manually provided attributes.
Activity 1 *Record detected date / time*	Detected date/time (not automatically logged, to be passed to activity 2)
Activity 2 Register to GNATS	Bug ID, bug description, date/time, priority, assigned developer, etc.
Activity 3 Check in *source code*	Bug ID

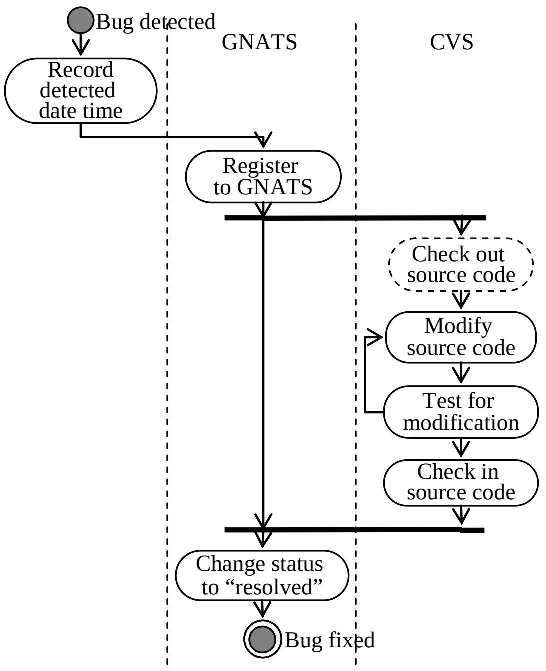


Fig. 4 Activity diagram for bug-fixing process model

omission of an activity (checking in modified source code). Fig 5 (c) is an incorrect sequence of checking in modified source code and changing status to "resolved".

3.3. Discussion

Through the example studies, we confirmed that micro process analysis can extract process instances and measure micro process metrics such as omissions of activity or incorrect sequence. Process instances in the study were extracted and visualized after the project had finished. However, process instances can be extracted and visualized during project is in progress. Extracting and visualizing process instances lead to increase visibility and opportunities to improve quality of process executions and logging especially in distributed software development. Quality of process executions is measured by followings:

- Omissions of activities
- Incorrect sequence of activities, and
- Invalid attributes

Quality of logging is inferred from followings:

- Inputs in a lump, and
- Missing inputs or operations (leave status unchanged)

Measuring execution time of each activity enables to detect potentially missing activities. For example, if duration from activity 3 exceeds twenty four hours, manager may notify developer that activity 4 is potentially omitted. Practically, actual notifications should be limited depending on risks caused by omission, because too many notifications are generally unwilling.

The micro process analysis evaluates not only quality of process instances but also feasibility of process model and procedure. For example, order violations between two activities activity1:"check-in source code" and activity2:"close bug track," suggest that bug-track is closed before the source code was checked-in. However, this many not be harmful if these activities happened in a short time,. Therefore, the manager may want to refine the model to accept such sequence as valid.

Though we believe that micro process analysis provides various valuable metrics at last, there are still some limitations and issues to be addressed. First, extraction of process instance doesn't always succeed and generally requires much computation. In the case study, missing attribute at activity 2 (bug ID) caused fails to extract process instance. Handling method for erroneous or ambiguous sequence should be considered for such cases. Also, the discussion of generic or specific algorithm for automatic extraction is required. Second, the accuracy of activity log should be carefully considered. Reflecting actual activity in activity logs depends on operation procedures of tools. When process models and operation procedures are defined, how far activity logs reflect actual activity should be taken into consideration. Collection environment that provides higher accuracy and better granularity for micro process analysis should be discussed.

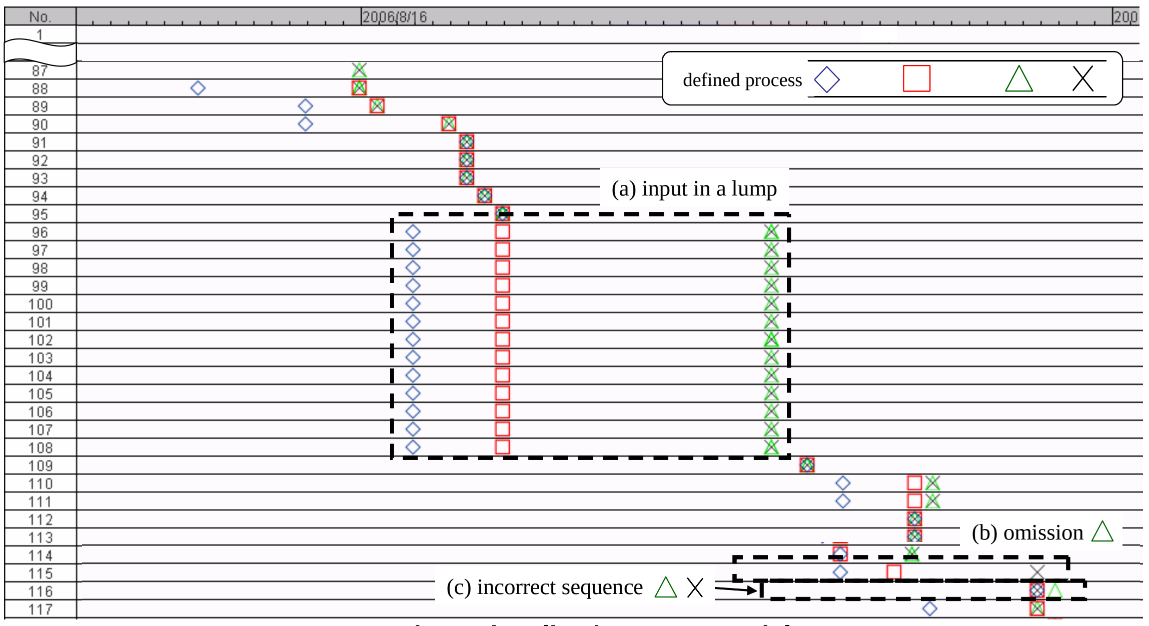


Fig. 5 Visualized process model

4. Conclusion

In this paper, we proposed a method for micro process analysis to distributed software development. Micro process analysis provides project manager with visibility of distributed sites. It also provides project manager with opportunities to ask development members to improve both quality of process executions and quality of logging, if necessary. As an empirical study, process instances are extracted from process model and activity logs collected in commercial software development. We confirmed that extracting and visualizing process instances potentially lead to identify issues in process executions by indicating an omission of activities and incorrect sequences of activities. We also confirmed that extracting and visualizing process instances potentially lead to identify issues in logging by indicating a chunk of input.

We are currently proceeding micro process analysis to a multi-vendored and distributed project to capture organizational characteristics from the view point of the process metrics. Future topics include evaluating the proposed method in ongoing project and further discussion for classification or limitation of process model and micro process analysis at different process model or different kind of activity logs.

Acknowledgement

Part of this research is supported by the Comprehensive Development of e-Society Foundation Software program of the Ministry of Education, Culture, Sports, Science and Technology of Japan. The authors would like to thank members of EASE project.

1. CMMI Product Team, “CMMI for Systems Engineering / Software Engineering / Integrated Product and Process Development / Supplier Sourcing. Version 1.2.,” CMU / SEI-2006-TR-008 (2006)
2. Froehlich J. and Dourish P., “Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams,” Proceedings of International Conference on Software Engineering, pp.387-396 (2004)
3. Johnson P. M., Kou H., Agustin J. M., Chan C., Moore C. A., Miglani J., Zhen S., Doane W. E.: "Beyond the Personal Software Process: Metrics collection and analysis for the differently disciplined", In Proceedings of the 2003 International Conference on Software Engineering, pp. 641 (2003)
4. Morisaki S., Matsumura T., Ookura K., Fushida K., Kawaguchi S. and Iida H., “Micro Process Analysis for Empirical Software Engineering Data,” 2006-SE-154-(2), Information Processing Society of Japan, pp. 9-15, (2006)
5. Ohira M., Yokomori R., Sakai M., Matsumoto K., Inoue K., Torii K., “Empirical Project Monitor: A Tool for Mining Multiple Project Data,” Proceedings of International Workshop on Mining Software Repositories, pp. 42-46, (2004)
6. Zimmermann T., Weisgerber P., Diehl S., and Zeller A., “Mining Version Histories to Guide Software Changes,” Proceedings of International Conference on Software Engineering, pp.563-572, (2004)