Software Qual J (2011) 19:101–120
DOI 10.1007/s11219-010-9105-8

ORIGINAL PAPER

Toward objective software process information: experiences from a case study

Jana Samalikova · Rob Kusters · Jos Trienekens · Ton Weijters · Paul Siemons

Published online: 24 August 2010

Abstract A critical problem in software development is the monitoring, control and improvement in the processes of software developers. Software processes are often not explicitly modeled, and manuals to support the development work contain abstract guidelines and procedures. Consequently, there are huge differences between 'actual' and 'official' processes: "the actual process is what you do, with all its omissions, mistakes, and oversights. The official process is what the book, i.e., a quality manual, says you are supposed to do" (Humphrey in A discipline for software engineering. Addison-Wesley, New York, 1995). Software developers lack support to identify, analyze and better understand their processes. Consequently, process improvements are often not based on an in-depth understanding of the 'actual' processes, but on organization-wide improvement programs or ad hoc initiatives of individual developers. In this paper, we show that, based on particular data from software development projects, the underlying software development processes can be extracted and that automatically more realistic process models can be constructed. This is called software process mining (Rubin et al. in Process mining framework for software processes. Software process dynamics and agility. Springer Berlin, Heidelberg, 2007). The goal of process mining is to better understand the development processes, to compare constructed process models with the 'official' guidelines and procedures in quality manuals and, subsequently, to improve development processes. This paper reports on process mining case studies in a large industrial company in The Netherlands. The subject of the process mining is a particular process: the change control board (CCB) process. The results of process mining are fed back to practice in order to subsequently improve the CCB process.

Keywords Software process mining · Configuration management data

J. Samalikova · R. Kusters · J. Trienekens (✉) · T. Weijters
University of Technology Eindhoven, Den Dolech 2, 5600 MB Eindhoven, The Netherlands
e-mail: j.j.m.trienekens@tue.nl

P. Siemons
Draugronth, Dwarsweg 3C13, 3959 AC Overberg, The Netherlands
e-mail: paul.siemons@draugronth.nl

1 Introduction

Software development is a discipline of increasing complexity, caused by both technology pushes and increased market and consumer needs for innovative software applications. Software development reflects a large variety of processes, of which many are difficult to define and consequently difficult to improve. Over the last decades, process assessment and improvement have become major topics in the software development domain (Humphrey 1995), (SEI 2006), and (Trienekens et al. 2009).

An important characteristic of software development is its mix of creative and routine-based development activities. Creative activities, such as functional and technical design, are often carried out in a flexible and unstructured way, ad hoc supported by computer-aided software engineering (CASE) methods, techniques and tools. Routine-based activities are carried out in a structured and repetitive way and are often supported by structured guidelines and procedures, which are specified in quality manuals. Examples of routine-based activities are peer reviewing, code testing and change request processing (SEI 2006).

Documents and files that are produced during software development are collected and stored in information systems, e.g., in software configuration management (SCM) systems. In these systems, also data on the development processes are stored, such as data on the tasks or activities carried out by the developers, and data on the creation and the changes on the documents and files. SCM systems act as a support for the monitoring and control of the development processes, e.g., to plan and coordinate the various development activities.

This paper will discuss the usage of data from SCM systems to analyze and improve a particular routine-based and repetitive software development process. Based on sets of well-prepared data, the underlying 'actual' processes will be extracted and process models will be constructed automatically. This is called software process mining (Rubin et al. 2007). Process mining has proven to be a valuable approach that provides new and objective insights into the way processes are actually carried out within organizations (Weijters et al. 2006). Process mining has been developed in domains with structured and less structured processes, as in hospitals where the data of all kinds of information systems are used. In software development, many creative processes are less structured, but in particular, the data of routine-based, i.e., structured and repetitive, processes are suitable for process mining.

The process that is subject to process mining in this paper is the control flow of a CCB in a large industrial organization in The Netherlands. This CCB is an organizational unit that handles change requests that are identified during software development. The control flow reflects the tasks and their dependency relations, which are carried out by the CCB. The 'official' CCB process is specified in the quality manuals of the company. After discovery and construction of the 'actual' process model, using process mining, this process model will be compared with the 'official' process, and the differences will be discussed with the software development teams. Based on the outcomes of these discussions, concrete process improvement actions can be determined.

This paper is organized as follows. In Section 2, a brief overview is given of the background and related research. Section 3 gives information on the software projects and the 'official' CCB process, such as specified in the quality manuals of the company that should be followed in these projects. After presenting the data available and the preparation of these data in Section 4, Section 5 will discuss the construction of 'actual' process models of the CCB process. Section 6 finalizes the paper with conclusions.

2 Background and related research

Software process improvement can currently be accomplished through various approaches, methods and techniques. A commonly accepted mainstream in software process improvement focuses on the assessment (Dorling 1993) and subsequent improvement in software development processes, e.g., capability maturity model integration (CMMI) (SEI 2006). Different SPI domains are recognized, such as a project management, engineering, a support and an organizational domain. In CMMI assessments, strengths and weaknesses of so-called key process areas are investigated by means of interviews and document studies. Subsequently, organization-wide improvement programs are determined (Trienekens et al. 2009). Although valuable results have been achieved in SPI, no explicit models of 'actual' development processes are being constructed. It is therefore questionable whether an in-depth understanding of particular 'actual' software processes can be achieved.

To model software processes explicitly, process mining offers interesting opportunities. Process mining has already been applied in different industrial domains (van der Aalst and Weijters 2005; van Beest and Maruster 2007). In the software industry, the behavior of development processes is being investigated from different points of view (Cook and Wolf 1998). More recent mining research has addressed the different types of process mining approaches and techniques (van der Aalst et al. 2003) and has resulted in a process mining framework for software processes (Rubin et al. 2007). Different aspects of processes have been addressed, such as the control aspect (capturing the order in which activities or tasks are executed), the information aspect (capturing the data, documents and information needed and produced by a task) and the organization aspect that captures which persons in which role execute a task.

Well-defined and structurally stored data about these process aspects act as a basis for process mining. To mine the different aspects of software development processes, different algorithms can be used. The ProM framework (van Dongen et al. 2005) offers a variety of process mining algorithms. ProM also provides interfaces to extract information from different sources, including SCM systems. Process mining algorithms can then be applied to discover the underlying processes of the available data and to construct automatically explicit process models. Depending on the mining goals, ProM offers algorithms for the mining of different aspects, e.g., control flow, resources, performance, organization. Further, a distinction can be made between ProM algorithms that focus on the main behavior of processes and are robust to exceptions and noise, and algorithms that focus on particular process details. Also, the verification of constructed process models is supported by process mining algorithms, captured in so-called plug-ins. Examples are the conformance checker and the performance sequence diagram plug-ins (Rozinat and van der Aalst 2008).

Table 1 is an example of a so-called event log. This event log contains information from a SCM system about the activities that have been performed by particular software developers in an 'actual' situation. Also, information on the start and the completion of activities, i.e., the event type, is given in the Table, and for each activity a timestamp.

Until recently, the information on these event logs was rarely used to analyze and construct 'actual' software processes. Originally, these data were used for e.g., the measurement of project activities, e.g., the amount of produced failures, and for the detection and prediction of changes in the code. However, in process mining, these data can be used to discover underlying software process models, to analyze, model and subsequently improve them. The event log in Table 1 is suitable for 'control flow-oriented' process mining, i.e., to discover the underlying process model that reflects the order in which the activities are executed. The information on the timestamp of an event and its originator

Table 1 Example of a event log

Case id	Activity	Event type	Originator	Time stamp
Case 1	Activity A	Start	John	9-3-2004:15.01
Case 2	Activity A	Start	John	9-3-2004:15.12
Case 3	Activity A	Start	Sue	9-3-2004:16.03
Case 3	Activity B	Start	Carol	9-3-2004:16.07
Case 1	Activity B	Start	Mike	9-3-2004:18.25
Case 1	Activity C	Start	John	10-3-2004:9.23
Case 2	Activity C	Start	Mike	10-3-2004:10.34
Case 4	Activity A	Start	Sue	10-3-2004:10.35
Case 2	Activity B	Start	John	10-3-2004:12.34
Case 2	Activity D	Start	Pete	10-3-2004:12.50
Case 5	Activity A	Start	Sue	10-3-2004:13.05
Case 4	Activity C	Start	Carol	11-3-2004:10.12
Case 1	Activity D	Start	Pete	11-3-2004:10.14
Case 3	Activity C	Start	Sue	11-3-2004:10.44
Case 3	Activity D	Start	Pete	11-3-2004:11.03
Case 4	Activity B	Start	Sue	11-3-2004:11.18
Case 5	Activity E	Start	Clare	11-3-2004:12.22
Case 5	Activity E	Complete	Clare	11-3-2004:13.17
Case 5	Activity D	Start	Clare	11-3-2004:14.34
Case 4	Activity D	Start	Pete	11-3-2004:15.56

(the person having triggered its occurrence) can be used to derive information about the underlying process also from other perspectives (Rubin et al. 2007).

From the control-flow perspective, the event log in Table 1 contains information about five cases. A case is a software component that follows a sequence of activities. Each activity is executed by an originator (a resource or (a set of) person(s) involved in a sequence of activities). This event log shows that for four cases (1, 2, 3 and 4), the activities A, B, C and D have been executed. For the fifth case, only three activities are executed: activities A, E and D. Each case starts with the execution of A and ends with the execution of D. If activity B is executed, then also activity C is executed. However, for some cases, activity C is executed before activity B.

Only for activity E, the start and complete events are registered. For all other activities, only the start event is registered. Based on the information shown in Table 1 and assuming that the cases are representative and a sufficiently large subset of cases is observed, process mining techniques can be used to construct a process model such as presented in Fig. 1.

The process starts with activity A and finishes with activity D. After executing A, there is a choice between either executing B and C in parallel or just executing activity E. Using the timestamp and resource (i.e., originator) information, it is possible to mine other process perspective such as performance and resources.

Different process model mining algorithms are available, and many of them are implemented in the ProM framework (van Dongen et al. 2005). As an illustration of the mining technique, we shortly discuss the ideas as implemented in the heuristic mining tool (the tool used in this paper). To find a process model on the basis of an event log, the log should be analyzed for causal dependencies, e.g., if an activity is always followed by another activity, it is likely that there is a causal relation between both activities.

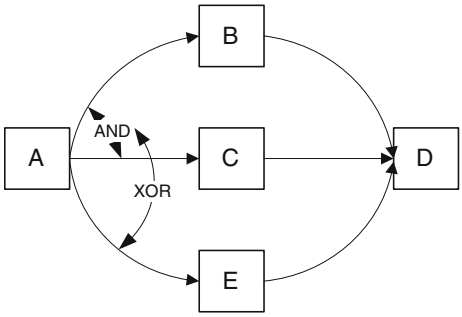

Fig. 1 An 'actual' process model as a result of process mining from the control-flow perspective

To analyze these relations, we introduce the so-called *direct following frequency metric* (notation #X > Y). Consider for example the event traces of the log of Table 1: ABCD; ACBD; ABCD; ACBD; AED. In this example, #A > B = 2 because there are 2 instances of A directly followed by B. The *dependency measurement* between two events X and Y (notation X → Y) is defined as (#X > Y − #Y > X)/(#X > Y + #Y > X + 1). In other words, the number of positive observation (#X > Y) minus the number of negative observations (#Y > X) divided by the number of observations plus 1. That means that in the example log A → B = (2 − 0)/(2 + 1) = 0.66. The intuition behind the plus 1 is to make the measurement sensitive for the number of positive observations. That is the reason that the dependency between A and B is relatively low (0.66). In a more realistic setting, event logs contain much more material and dependencies will get values close to 1 (see also the discovered CCB process model in Sect. 5.1). However, realistic event logs can also contain some errors. In (Weijters et al. 2006), it is illustrated how the dependency measurement in combination with some heuristics can be used to construct a complete process model with Split/Join information. In Fig. 1, the mining result of the heuristic mining algorithm for the event log of Table 1 is presented.

In many real-life development situations, event logs with process information as in the foregoing example, see Fig. 1, are not directly available. Event logs often contain either too many details or very specific information on different aspects of software processes (Rubin et al. 2007). However, it often is possible to combine information from different sources to construct useful event logs. In order to achieve a useful data set, data preparation has to be carried out, in that the quality of the available data has to be examined and improved (Witten and Frank 2005). After presenting some background information about the case study and the 'actual' and 'official' CCB process in the next section, the data preparation will be discussed in Sect. 4.

3 The case studies: software projects and their CCB process

The projects under study are middleware embedded-software projects of an industrial company in the Netherlands. The company develops software components for consumer electronic devices, which are going to be released in the near future. Over the past years, the company reached level 3 of the CMMI (SEI 2006). This means that the organization is capable of defining its software development processes and interrelated activities. On this level of maturity, these activities can be specified, examined and measured, and data can be collected and stored in a structured and accurate way. This kind of data offers opportunities for the application of process mining.

3.1 The complexity of the 'actual' CCB process

The 'actual' CCB process in ten software projects will be discovered and analyzed. The different types of updates of the software components, called releases and versions, make the software development processes in the projects very complex. For instance, a previous version of a device does not have to be necessarily completed in order to start the development of a next version. In addition, the development activities for the different releases and versions are often run in parallel. SCM systems are used to keep track of all the software components and their versions. The components and versions are called configuration items (CIs). The migration of a CI from one software version to the next one in the software development process is based on change requests. The CCB analyzes these change requests and tracks them in order to monitor their status, to plan necessary activities in the project and to predict outcomes of the development processes. The product and process complexity of the projects stresses the high importance of an efficient and effective CCB process. To discuss the application of process mining in detail, one of the projects (called project P) is selected. In the following section, the 'official' CCB process will be presented that is used in the software development projects. This 'official' process is derived from the guidelines and procedures in the quality manuals of the company.

3.2 The 'official' CCB process as derived from the quality manuals

The CCB coordinates changes made to the CIs. The CCB tracks and records the status of each change request from its entry until its exit from the CCB process. The change requests are further referred to as defects. The responsible departments for requirements engineering and programming carry out the tasks in this process. The role of the CCB is to distribute tasks related to the required change of the CIs and evaluate the outcomes of the executed tasks with respect to the request.

The structure of the CCB process is sequential with possible rework if a task fails, see Fig. 2. The tasks are not executed in parallel, and each task is completed before the next task starts.

The flow of tasks of the CCB process is as follows:

Task 1. The CI's defect is detected and submitted. The developer assigns attributes to the defect (e.g., priority, severity). Based on the importance, the defect is either:

- A. further evaluated by the CCB (Task 2).
- B. or the defect will directly start with the *Analysis* task (Task 3).

Task 2. The CCB analyzes the defect and sends it to the required task depending on the need (*Analysis, Resolution, Evaluation* or *Concluded* task), with the following possibilities:

- C. The defect is redirected to the *Concluded* task in case the defect is found duplicated, expected to be fixed in a next release, or out of the scope of the functionality required.
- D. The defect is redirected to tasks *Analysis*, *Resolution* or *Evaluation* depending on the need.

Task 3. The task, i.e., *Analysis*, *Resolution* or *Evaluation*, starts to handle the CIs. When the task is completed, one of the four possibilities is chosen:

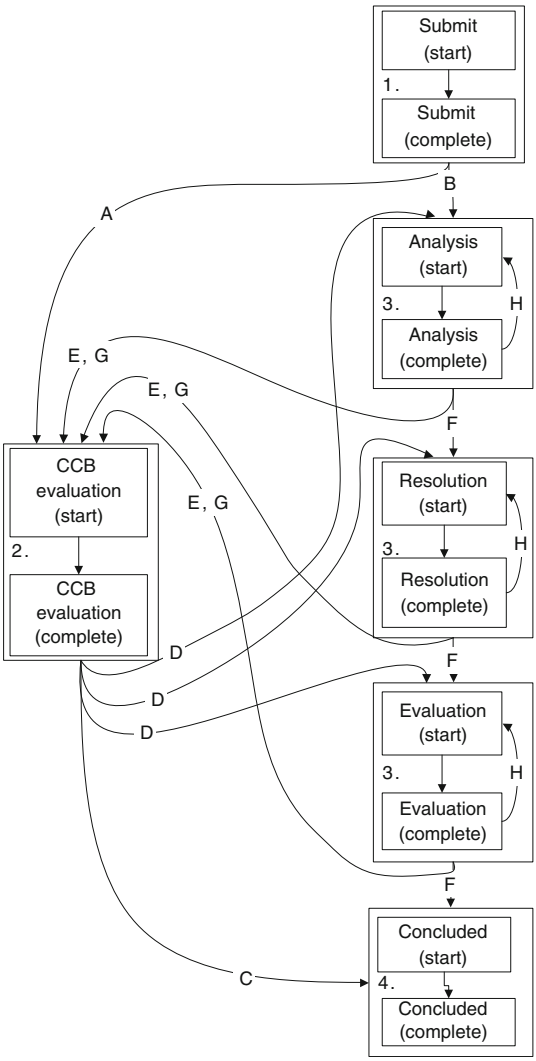

Fig. 2 The 'official' CCB process model

E. If the task's execution is successful, then an important defect is directed to the CCB and it waits to be redirected again to the next task (it returns to Task 2).
F. If the task's execution is successful, then a less important defect continues with the next logical task, for instance after *Analysis* it can be *Resolution*.
G. If the task was not successfully executed, then an important defect is returned to the CCB for a re-evaluation (Task 2).
H. If the task was not successfully executed, then a less important defect is handled again by the same task (Task 3).

Task 4. Once all the tasks of the CCB process have been successfully carried out, the pattern of the defect is closed.

Based on the quality manual of the company X, the tasks of the CCB process are described as follows:

3.2.1 Submit task

The task is performed by a tester. During the task, a defect is submitted and it receives an identification number in the CCB system.

3.2.2 CCB Evaluation task

The task is performed by the change control board (CCB). During the task, the CCB evaluates properties of the defect (e.g., severity, priority). Based on these properties, a pre-selection of defects is made and a decision of the next steps is taken. Also, results of the tasks Analysis, Resolution and Evaluation are analyzed with respect to major defects.

3.2.3 Analysis task

The task is performed by an analyst assigned by the CCB. During the task, a solution for the request submitted is identified. This includes reconstruction of the problem for a problem report, proposed technical solution (with possible alternatives when applicable) for the change request or problem report, estimation of the change impact on the project and the system and identification of all components affected by the handled defect.

3.2.4 Resolution task

The task is performed by a coder or programmer. The defect is being resolved based on the solution identified during Analysis. The programmer also ensures that all relevant documentation and code are updated accordingly.

3.2.5 Evaluation task

The task is performed by a verifier. During the task, outcomes of the Resolution task are evaluated with respect to the change requested. Also, a decision is made whether the handled defect needs rework (e.g., because the implementation is incomplete or incorrect) or the handling of the defect is complete.

3.2.6 Concluded task

The task is performed by the CCB. During the task, the defect is closed and the final status of the defect is reported to the original initiator. The changed documentation and code are correctly archived in the repository.

The process model is shown in Fig. 2, where labels are assigned based on the foregoing description. The numbers represent the tasks; the characters A to H represent the transitions between the tasks.

This ‘official’ CCB process model will be compared with ‘actual’ CCB process models. Regarding this ‘actual’ CCB process model, the data preparation will be described in Sect. 4, and the analysis and the construction of this ‘actual’ CCB process model will be presented in Sect. 5.

4 The case studies: the available data and the data preparation

The development team of the project P collects data in a status database in order to be able to control the development process and to predict its outcomes. A quality assurance specialist creates copies of the status database content on a weekly basis. Such a copy is called a *snapshot* of the CCB database. The snapshots provide the data for the creation of an event log, which serves as input to the process mining. One of the research questions was whether it is possible to use such data to discover and construct underlying process models, since the quality of the data is crucial for a successful application of process mining. In the next section, we describe the transformation and cleaning process from the snapshots of the CCB database into an event log. Notice that this transformation and cleaning process can take 60–80% of the time of the whole mining process (Witten and Frank 2005).

4.1 The available data

As indicated before, information about a defect and its status during the CCB process is stored in a database of the SCM system. The defects have been discovered during verification, and validation activities. Every database record describes the defect by its attributes and timestamps. The snapshots contain the information of the SCM system on a weekly basis. Table 2 shows three example snapshots and their changes over a period of 3 months.

The snapshots follow the evolution of the handling of the defects by the CCB. The evolution is captured in the field *CrStatus* that stores the information of the current status of a particular defect. Each snapshot contains a record for each defect that is described by four types of data fields:

1 The *History Date* and *Subsystem* data fields provide the general reference about the snapshot; they describe the date of the snapshot and the subsystem database from which the snapshot was taken.
2 *Problem_nr* together with the *Subsystem* data field uniquely identifies the defect.
3 The *Priority*, *Severity*, *Request_type*, *CrStatus* and *Team* fields describe the attributes of the defect.
4 The dates of *start* and *complete* are stored in the corresponding fields (e.g., the *start* event of the *Analysis* is stored in *Analysis (started)* field) and the *Modify time* field stores the date of the last change of the CI's status.

4.2 Data preparation: from snapshots to event log

The snapshots capture the evolution of the defects, i.e., the changes of the defect status, in time. This information has been used as the input for the mining of the underlying 'actual' process that handles the defects. The first step is the transformation of the available data into an event log. Then, the event log is used as an input for process mining. In this section, we describe the necessary transformation of the data fields in the snapshots into an event log. First, we start with a description of the data structure of an event log.

Data structure of an event log:

- *Case identifier*. Cases (or instances) are items that are handled by a process [1]. Here, a defect is considered as a case, and the *Problem_nr* is the case identifier.

Table 2 An example of snapshot records of Defect nr. 2714

Data field	Value
(a) August 22, 2007	
History date	22-08-07
Subsystem	SUB1
Problem nr	2714
Priority	Medium
Severity	B
Request type	PR
CrStatus	In_resolution
Team	TEAM1
Submit (start)	04-09-06
Submit (complete)	04-09-06
Analysis (start)	18-01-07
Analysis (complete)	27-03-07
Resolution (start)	02-04-07
Resolution (complete)	
Evaluation (start)	
Evaluation (complete)	
Modify time	02-05-07
(b) September 26, 2007	
History date	26-09-07
Subsystem	SUB1
Problem nr	2714
Priority	Medium
Severity	B
Request type	PR
CrStatus	In_resolution
Team	TEAM1
Submit (start)	04-09-06
Submit (complete)	04-09-06
Analysis (start)	18-01-07
Analysis (complete)	27-03-07
Resolution (start)	02-04-07
Resolution (complete)	
Evaluation (start)	
Evaluation (complete)	
Modify time	05-09-07
(c) October 22, 2007	
History date	22-10-07
Subsystem	SUB1
Problem nr	2714
Priority	Medium
Severity	B
Request type	PR
CrStatus	Concluded

Table 2 continued

Data field	Value
Team	TEAM1
Submit (start)	04-09-06
Submit (complete)	04-09-06
Analysis (start)	18-01-07
Analysis (complete)	27-03-07
Resolution (start)	02-04-07
Resolution (complete)	17-10-07
Evaluation (start)	
Evaluation (complete)	
Modify time	19-10-07

- *Tasks* are executed when they handle a case during a process. Tasks have been derived from the *CrStatus* field. A change in the status of a task is an event. In the snapshots, only the event types *start* and *complete* are used. Possible other event types such as *suspend* or *resume* are not used in the snapshots.
- *Timestamps* are points in time of each event that is executed during the handling of a case in a process. The necessary timestamps have been extracted from the fields in the snapshots that store time information.
- *Resources* are persons that are involved in the execution of each task during the handling of a case in the process. This information has been derived from the *Team* field in the snapshots.
- *Case-related attributes* are attributes that can enrich mined process models and/or can play a role in process mining from different perspectives. Regarding the CCB process of project P, the defined case-related attributes are derived from the *Priority, Severity* and *Request_type* fields.

At first sight, it seems relatively simple to transform the snapshots into a corresponding event log. However, a number of problems occurred. The following problems were detected: missing fields in records, incorrect event sequences and absent information about tasks and events. Some of these problems are caused by the possibility to overwrite and/or delete information in the SCM system. For instance, each *Submit, Analysis*, *Evaluation* and *Resolution* status change has a timestamp assigned to its *start* and *complete* event, see Table 2. Any status change is a result of an event. At each *start* event of such a task, the corresponding data field in the database is filled in. When the execution of the task is successful, also the timestamp of the *complete* event is recorded. In the case of a failure of a task, the corresponding timestamp of the *start* event is removed. For example, if the execution of the *Analysis* task was not successful, the data field *Analysis (started)* is set to be empty. Depending on the moment snapshots are taken, this can for instance result in an incomplete event pattern like Analysis (start), Analysis (start), Analysis (complete). A possible explanation is the following snapshots collection: Analysis (start) [snapshot 1], Analysis (complete) [missing], Analysis (start) [snapshot 2], Analysis (complete) [snapshot 3]. As indicated, the following problems were detected: missing fields in records, incorrect event sequences and absent information about tasks and events. These three problems have been dealt with as follows.

4.2.1 Problem 1: missing fields in records

The snapshots contain the cumulative defects submitted to the CCB. However, records in snapshots have been identified, which contained these cumulative numbers, while subsequent snapshots did not have these numbers. To improve the quality of the data, such records have been excluded from the analysis, as they were apparently considered not to be defects.

4.2.2 Problem 2: incorrect sequences of events

Table 3 lists the incorrect sequences together with the (simple) strategies, which have been used to correct them. Missing *start* and *complete* events have been introduced to enhance the completeness of the data. It has to be emphasized that these strategies are only possible due to the known fact that the activities within the CCB process do not run in parallel and that a previous task is completed before the next one starts.

When performing the resolving steps described in Table 3, also artificial timestamps have been assigned to newly introduced events. As an event for a task is recorded, the task was actually executed. Hence, the artificial events of such tasks do not create any new tasks that were not performed. In this case, no start and no complete event of a task was recorded, and the task was not executed (i.e., skipped). For the sequences described in

Table 3 Possible incorrect sequences of events and strategies for fixing these situations

Incorrect sequence	Resolving strategy	Number of inserted events
Two consecutive start events coming from different tasks, e.g., A and B. Example: *A (start), B (start)*	An artificial *complete* event has been introduced between both start events. The artificial event will belong to the task of the first start event. Example: *A (start), A (complete), B (start)*	2
Two consecutive *start* start events coming from the same task but having different time stamps. Example: *A (start), A (start)*	Both events have been considered as two separate executions of the same task, and the same strategy has been applied as in the previous case, i.e., we introduced an artificial complete event between them. Example: *A (start), A (complete), A (start)*	12
Two consecutive *complete* events coming from different tasks. Example: *A (complete), B (complete)*	An artificial *start* event has been introduced between the two *complete* events. The artificial event will belong to the task of the second complete event. Example: *A (complete), B (start), B (complete)*	608
Two consecutive complete events coming from equal tasks but having different time stamps. Example: *A (complete), A (complete)*	Both events are being considered as two separate executions of the same task and introduced an artificial start event between the two *complete* events. Example: *A (complete), A (start), A (complete)*	0
The start event of a task followed by the complete event from a different task. Example: *A (start), B (complete)*	Two artificial events have been introduced: first, the *complete* event of the first task and then the *start* event of the second task. Example: *A (start), A (complete), B (start), B (complete)*	0

the first four rows in Table 3, the timestamps from the timestamp of the previous event have been calculated with plus 30 min. As for the cases described in the last row of the table, the timestamp of the artificial *complete* event has been calculated from the timestamp of the *start* event plus 20 min and the timestamp of the artificial *start* event plus 40 min. The problems and the solutions were evaluated with the quality assurance specialist. Although the throughput time of a case is not affected by the introduction of artificial time stamps, the disadvantage of artificial timestamps is that a detailed performance analysis (e.g., process bottlenecks identification, execution and waiting times, etc.) may become unreliable. However, a detailed performance analysis is out of the scope of this paper.

4.2.3 Problem 3: absent information about tasks and events

Since the snapshots do not log processes, the data miss explicit information about performed tasks and events. The change in a defect's status is a result of an activity performed on the defect. Hence, tasks have been derived from the current status of a particular defect in a snapshot (i.e., from the *CrStatus* data field in the snapshot). The identification of the tasks *Submit, Analysis, Resolution and Evaluation* is straightforward. A defect is considered to be handled by the *CCB evaluation* tasks when the *CrStatus* of the defect is one of the following: *Duplicate, On Hold, Later Release, Not reproducible* or *Rejected* (as these status values are only assigned during the *CCB evaluation)*. The *Concluded* task is identified when the *CrStatus* is equal to *Concluded.* The *Modify time* field has been used for retrieving the *Concluded* and *CCB evaluation* task's timestamp, see Table 2. Unsuccessful executions of the task *Analysis, Resolution* and *Evaluation* are respectively recorded as *Analysis failed, Resolution failed* and *Evaluation failed* values of the *CrStatus* field.

In making these changes, and indeed in the entire process mining approach, we assume that the activities as carried out in practice can be mapped in a reasonable fashion to the data available. This firstly implies that no other type of activities are carried out that are relevant for the CCB process. Given the long experience with this type of process, this seems a reasonable assumption. A second assumption is that these activities are carried out as specified. This is a less reasonable assumption. One can easily imagine that during analysis, the solution is identified and found to be so obvious that the resolution task is carried out straight away. In the data, this can then be recorded by a complete data sequence, but it also could explain some of the missing data. For instance, a sequence "A (start), B (complete)" could indicate such a case where the start of analysis and the completion of the resolution are entered and the intermediate data are kept blank since they are perceived to be not relevant. However, this almost never occurred in the data. Similarly, the sequence "A (start), A (complete), B (complete)" can refer to such a situation where the start of the resolution activity is noted, but the not so relevant information on the completion of the analysis task is left out. In fact, in the data, we do find that the second sequence occurred often (see Table 3). This can indicate a normal omission in data recording, but it can also indicate a case where sequentially defined activities are carried out jointly. The techniques we used are incapable of identifying which is true, but the analysis of missing data does suggest that this deviation between prescribed and executed process could occur. Validation of this suspicion requires checking with the developers. In the remainder of the analysis, we will not focus on this potential problem and focus on what can be learned from the cleaned data.

After correction, it was possible to identify 8,832 cases in total. After filtering out the cases which do not start with a Submit task, and the cases which do not end with a

Concluded task, an event log with 6,870 cases remains. This event log has been used to mine the process model of the 'actual' CCB process.

5 Analyzing and constructing the 'actual' CCB process

For discovering the 'actual' CCB process model from the event log, process mining algorithms from ProM 5.0 have been used (van Dongen et al. 2005). This 'actual' process model is compared with the 'official' CCB process model. The differences between these models are discussed with the development teams, and subsequently, improvement actions are determined.

5.1 The discovered 'actual' CCB process model of project P

The goal of process mining in the project P was to use the event log with the 6,870 completed cases to discover the 'actual' way of working, i.e., the control flow, of the software developers in the CCB process. Based on the data that have been used and the problems identified in Sect. 4, the heuristic mining algorithm of ProM has been selected to mine the underlying CCB process from a control-flow perspective. This heuristic algorithm is relatively robust and has options to focus on the main behavior of a process, instead of trying to model the full details of the behavior of a process (Weijters et al. 2006).

Figure 3 presents the discovered 'actual' CCB process model that has been constructed by using the heuristic algorithm of ProM. Two numbers label each transition from one task to another. The upper number in the figure describes the reliability of the transition. The reliability scale goes from 0 to 1, where 1 represents the highest reliability. It has to be noticed that the transition becomes more reliable when the transaction is support by more cases (i.e., if only one case shows this behavior, the reliability is very low). The lower number describes the number of cases that have passed the transition. For example, the transition from *Submit* to *Analysis* is reliable (1), and 2,081 cases have passed this transition, see Fig. 3.

By using the default parameters of the heuristic mining algorithm, only the main dependency relations are presented. It has to be emphasized that the information can be incomplete because the available data, i.e., the snapshots, were taken on a weekly basis.

The following differences between the control-flow paths in the 'actual' CCB process model, in comparison with the 'official' CCB process, have been identified. The first one is the (illegal) direct transfer of a case from the tasks *Analysis* to *Evaluation* that was followed by 0.74% (51) of the cases. The second one, and definitely more important, is the (also illegal) transfer from the tasks *Submit* to *Resolution,* without passing the task *Analysis.* This skipping of the task Analysis was followed by 70% (4,779) of the cases. This was confirmed by a conformance check. The aim of conformance checking (Rozinat and van der Aalst 2008) is to test how much of the behavior captured in the event log (see Sect. 4; Fig. 3) is in compliance with a process model. We use this technique to compare the event log with the 'official' CCB process model, see Fig. 2. In other words, the conformance checking detects mismatches between the discovered process model and the logged execution of the process such as expressed in the event log. The result was that only 2,035 out of the 6,870 cases (i.e., 30%) in the event log were fully compliant with the defined 'official' CCB process.

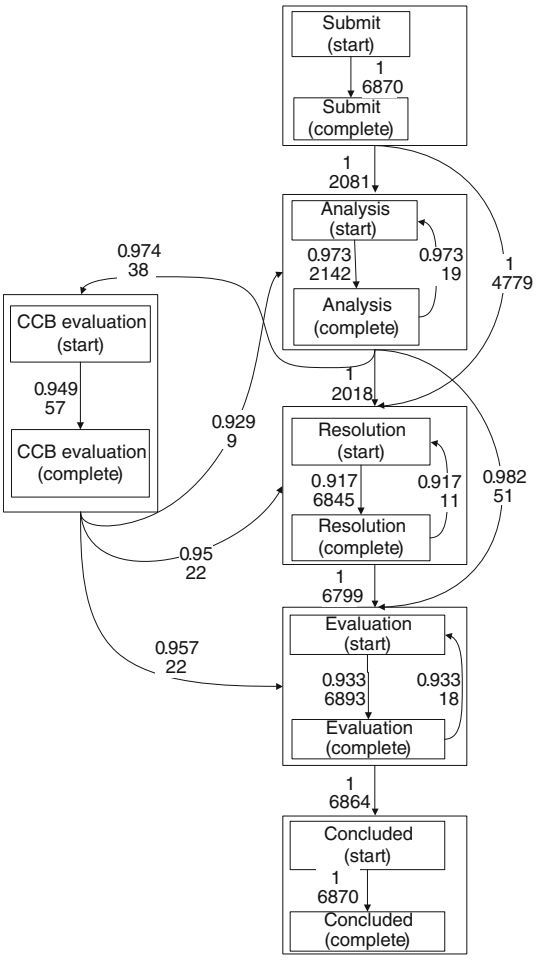

Fig. 3 The 'actual' CCB process model discovered with the ProM heuristic mining algorithm

Since the *Analysis* task is considered as one of the most important tasks in the CCB process, skipping this task to this extent is very surprising, and further analysis of this phenomenon is urgently needed.

5.2 Control-flow patterns

The event log has also been analyzed using the *Performance Sequence Diagram* algorithm (or ProM plug-in). This plug-in provides information about what sequences of activities, i.e., patterns, in the process are common and what sequences are less frequent. The analysis shows that although there are 45 different sequences, most of the behavior of the cases (6,699 or 97.5%) can be described by two (most common) sequences. The illegal sequence 1 is described by the sequence *Submit* → *Resolution* → *Evaluation* → *Concluded* and covers 4,742 cases. Sequence 2, which is legal, describes the behavior of handling 1,957 cases in the following sequence: *Submit* → *Analysis* → *Resolution* → *Evaluation* → *Concluded*.

5.3 Feedback to the development team on the results of project P

After discovering the 'real' process model using process mining in the previous sections, several differences between the mined and documented model were revealed. These differences could result from various situations. The type of the situation is not recognized automatically; it requires involvement of a process owner. That means that process mining results are a starting point for further analysis and must be fed back to the development team. Hence, it is not possible to make a decision regarding the process without understanding circumstances that may influence its execution and cause the differences between the documented and the mined process model. Based on such analysis, it is decided whether the difference is a result of an exceptional or systematic behavior and which of the models—documented or mined—is 'wrong'. This discussion, however, requires further research, and it is beyond the scope of this paper.

In particular, the main result of the process mining, i.e., the discovery of the skipping of the task Analysis, has been discussed with the development team of project P. The high percentage (70%) of cases following the path from the task *Submit* directly to the task *Resolution*, skipping *Analysis*, could indicate that this is rather common behavior not an exception.

The responsible manager of the project P explained at first that the tight schedule and not having enough managerial commitment to follow the 'official' CCB process played a role in allowing deviations from that 'official' process. Project P was a fixed price project, and due to a slow start-up, it had 'wasted' a significant part of its budget. For that reason, it was decided to 'ease' on 'official' CCB tasks where these would not influence the final quality or the timely delivery (to be decided by the developers themselves). Furthermore, it was argued that project P was transferred from a CMM level 3 organization to a joint venture with an external development party.

A more in-depth analysis of the cases that skipped the task *Analysis* with the project team showed that almost half of these cases (2,123) were so-called implementation requests. That means that these cases were not defects in terms of errors, but implementation requests for various functional specifications. Such implementation requests may not require analysis and may be resolved directly. However, this is not explicitly mentioned in the 'official' CCB process description. Regarding the other half of the cases, it appeared that the task *Analysis* was usually skipped if a defect was considered to be 'simple' and the solution of it to be straightforward. This behavior appeared to be allowed for particular circumstances and under certain conditions (also to be decided by the developers themselves). These particular circumstances and certain conditions were not explicitly mentioned in the 'official' CCB process.

The researchers finally suggested to the development team that it should include these circumstances and conditions in the 'official' CCB process description (i.e., in the quality manual) and as such to improve the 'official' process on the basis of the mined 'actual' CCB process model.

A possible occurrence of joint task execution has also been discussed with developers. In case of straightforward solutions, a defect was indeed sometimes resolved without completing the Analysis task first. That means, a number of defects followed an unidentified process path. In that case, the defects were processed during a new concatenated task. Even though concatenating Analysis and Resolution was not defined in the quality manual, such behavior was not necessarily considered to be illegal. Non-critical defects whose solution is considered to be simple and straightforward could be resolved immediately. The approach was proposed to reduce bureaucracy during the development process. Nevertheless, it would be recommended to use this scenario consciously and only if certain

conditions are met. These conditions then also need to be explicitly defined in the quality manual.

To verify the information on the specific characteristics of project P, which had been received from the project team, it was decided to investigate the event logs of nine other projects in the same software development organization. In particular, the skipping of the task *Analysis* in the 'actual' CCB processes of these projects has been investigated.

5.4 Projects P1 to P9: mining results regarding the skipping of the task *Analysis*

In the projects P1 to P9, the same 'official' CCB process from the quality manuals has been used by the developers. For each project, the available data have been prepared similar to the data preparation in project P, leading to nine separate event logs. Process mining was done on these event logs for each of the projects separately and not on the data set as a whole, as the goal was not to get insight into 'one overall actual' CCB process, but to provide feedback on the differences between the 'actual' CCB processes to the distinct development teams.

Also, in the nine projects, the heuristic miner plug-in of ProM has been used to discover the 'actual' CCB process models. The results are similar to the main result of the mining of the event log from project P. In each of the projects P1 to P9, it appeared that the *Analysis* task was NOT executed in a considerable amount of the cases. In Table 4, this main mining result is presented.

5.5 Feedback to the development team on the results of the nine projects

As can be seen in Table 4, the skipping of the task *Analysis* takes place often. In total, it occurs in 50% of cases. Apparently, project P was not such a special project after all, and probably, a more fundamental cause for the deviations from the 'official' CCB path has to be identified. Confronting the development teams with these data resulted in an acceptance of the seriousness of the deviation, the necessity to do further research on the mining of the CCB processes and to subsequently improve the 'official' CCB process of the company.

It is interesting that it took the mining results of more than one, i.e., nine, project to achieve an acceptance of the main mining result by the development team of project P. The mining results from the single project P were at first, more or less, discarded without much reflection, since the project was 'special', and carried out under specific circumstances. Convincing the

Table 4 Percentages of cases that skipped *Analysis* in the projects P1–P9

Project	Number of cases	*Analysis* skipped (%)
P	6,870	70
P1	343	33.82
P2	40,255	49.07
P3	10	10.00
P4	2,215	4.33
P5	195	32.82
P6	1,046	69.89
P7	2,228	33.39
P8	268	76.49
P9	785	79.62

development team with the same kind of results from nine other projects indicates that the discovery of structural process deviations requires data from several projects.

6 Conclusions

In this paper, we presented a case study on the practical application of software process mining in an industrial company. The subject for process mining in this case study was the 'actual' CCB process. Data collected and stored in ten software development projects have been used. These data capture the changing status of defects as they are handled by a CCB. The CCB process is 'officially' described in quality manuals of the company.

Process mining strongly depends on the quality of collected and stored data, and a quite large number of process instances in the available data sets had to be filtered out due to incompleteness. However, the paper shows that a careful data preparation can lead to useful event logs for the mining of processes, despite the fact that the data were originally not collected and stored to carry out process mining. Still, the efficiency and effectiveness of process mining can substantially benefit from well-structured data definition and collection guidelines and thus high-quality data sets.

The 'actual' CCB process models could be successfully discovered from the event logs using process mining algorithms. A second important result of this research is that a 'structural' deviation could be identified in the 'actual' CCB process models, in comparison with the 'official' CCB process. This deviation is the skipping, in a high number of cases (70%), of an important task in the 'official' CCB process, i.e., the task *Analysis.* Initially, this deviation has been identified in a case study on a project P that has been addressed first in this paper. This deviation was then discarded by the development team because of the exceptional project characteristics of that project. After carrying out the same type of process mining on the (prepared) data of nine other software development projects, it appeared that the same deviation as in project P was found. As a consequence, the deviation was called 'structural' and accepted by the project management as being a serious problem. Consequently, it has been decided to study this deviation further in order to improve the 'official' CCB process in the quality manual and/or to improve the 'actual' way of working in the CCB process. The paper has shown that detailed and well-founded software process improvements can be based on the results of analyzing and constructing explicit process models, i.e., on the results of process mining.

References

Beest van N. R. T., & Maruster L. (2007). A process mining approach to redesign business processes—a case study in gas industry. Symbolic and numeric algorithms for scientific computing, 2007. In *SYNASC. international symposium on* (pp. 541–548).

Cook, J. E., & Wolf, A. L. (1998). Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology, 7*(3), 215–249.

Dongen van B. F., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. J. M. M., & van der Aalst, W. M. P. (2005). The ProM framework: A new era in process mining tool support. *Applications and Theory of Petri Nets*, 444–454. In *Lecture Notes in Computer Science*. Heidelberg: Springer Berlin.

Dorling, A. (1993). SPICE: Software process improvement and capability determination. *Software Quality Journal, 2*(4), 209–224.

Humphrey, W. S. (1995). *A discipline for software engineering*. New York: Addison-Wesley.
Rozinat, A., & van der Aalst, W. M. P. (2008). Conformance checking of processes based on monitoring real behavior. *Information Systems, 33*(1), 64–95.
Rubin V, Gunther C, van der Aalst WMP, Kindler E, van Dongen BF, & Schäfer W. (2007). Process mining framework for software processes. Software process dynamics and agility, 169–181. Heidelberg: Springer Berlin. In *Lecture notes in computer science*.
SEI, CMMI Product Team. (2006). CMMI for development, version 1.2. CMU/SEI-2006-TR-008.
Trienekens, J. J. M., Kusters, R. J., Kriek, D., & Siemons, P. (2009). Entropy based software process improvement. *Software Quality Journal, 17*(3), 231–243.
van der Aalst, W. M. P., van Dongen, B. F., Herbst, J., Maruster, L., Schimm, G., & Weijters, A. J. M. M. (2003). Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering, 47*(2), 237–267.
van der Aalst, W. M. P., & Weijters, A. J. M. M. (2005). Process mining. In M. Dumas, W. van der Aalst, & A. ter Hofstede (Eds.), *Process-aware information systems* (pp. 235–254). Hoboken, NJ: Wiley.
Weijters, A. J. M. M., van der Aalst, W. M. P., & Alves de Medeiros, A. K. (2006). Process mining with the heuristics miner algorithm. BETA Working Paper Series, WP 166, Eindhoven University of Technology, Eindhoven.
Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed.). San Fransisco, CA: Morgan Kaufmann Publishers.

Author Biographies

Jana Samalikova is a PhD student within at the Industrial Engineering & Innovation Sciences department at the Eindhoven University of Technology. She focuses her research on software process discovery using process mining. Jana obtained her Master's degree at the University of Economics in Bratislava at the Information Systems department. After finishing her university degree, she joined the post-master program Software Technology at the Computer Science department at the Eindhoven University of Technology. During the program, she worked on several industrial research and development projects for Philips organization concentrating her work on the quality aspect of developing embedded software.

Rob Kusters (1957) obtained his Master's degree in econometrics at the Catholic University of Brabant in 1982 and his PhD in operations management at Eindhoven University of Technology in 1988. He is professor of 'ICT and Business Processes' at the Dutch Open University in Heerlen where he is responsible for the master program 'Business process Management and IT'. He is also an associate professor of 'IT Enabled Business Process Redesign' at Eindhoven University of Technology where he is responsible for a section of the program in management engineering. He published over 90 papers in international journals and conference proceedings and coauthored six books. Research interests include process performance, enterprise modeling, software quality and software management.

Jos Trienekens (1952) is an Associate Professor at TU Eindhoven (University of Technology—Eindhoven) in the area of ICT systems development. He is responsible for a research program on ICT-driven business performance and is an associate member of the research school BETA at TUE that focuses on operation management issues. Jos Trienekens published over the last ten years various books, papers in international journals and conference proceedings in the domains of software quality and software process improvement. He joined several international conferences as PC member and member of the organization committee. He is also an experienced project partner in various European projects.

Ton Weijters is associate professor at the school of Industrial Engineering of the Eindhoven University of Technology (TU/e) and member of the BETA research group. His current research focuses on data and process mining (i.e. to extract knowledge from event logs recorded by an information system to analyze the underlying business processes). He is the auteur of more than hundred scientific publications in the mentioned research field. His papers appeared in journals such as Data Mining and Knowledge Discovering, Information Systems, IEEE Transactions on Knowledge and Data Engineering, Artificial Intelligence in Medicine, European Journal of Operational Research, Computers in Industry, Knowledge-Based Systems, AI-Review, etc.

Paul Siemons has a very broad interest in science, technology and the arts. After studying physics, math and computer science, he started his career in the field of technical automation in the areas of engineering, project management and process improvement. As an improvement consultant, he is experienced in controlling process performance, and he has expertise in estimation, measurement and analysis. Over the years, he switched from management to consulting and founded his own company Metrific Management Consult in 2002. Since January 1, 2007, Paul is operating from Draugronth BV. Paul also actively participates in scientific research performed by several university departments and is coauthoring publications on measurement- and improvement-related subjects.