Activity Mining for Discovering Software Process Models

Ekkart Kindler, Vladimir Rubin, Wilhelm Schäfer
Software Engineering Group, University of Paderborn, Germany
[kindler, vroubine, wilhelm]@uni-paderborn.de

Abstract: Current enterprises spend much effort in obtaining precise models of their software engineering processes in order to improve the process capability of their organization and to move one step forward in the CMM. The manual design of process models, however, is complicated, time-consuming, error-prone and the results are rapidly becoming out-dated; capabilities of humans in detecting discrepancies between the actual process and its model are limited. Thus, automatic techniques for deriving process models are very important.

In this paper, we present an approach that exploits the user interaction with a document version management system for the automatic derivation of process models

1 Introduction

Obtaining precise descriptive models of the software engineering processes is an important task for many enterprises. Today, process engineers and managers are solving this problem manually. This is complicated, time-consuming, error-prone and the results are rapidly becoming obsolete. In addition, one usually does not start from scratch when defining an explicit process model; rather existing more or less informally executed processes must be taken into account. Therefore, tool support is needed for deriving the process models. Since a *document version management system* is an essential part in today's software engineering environment even when no precise definitions of the processes exist, we suggest using it as a source of input information for an automatic approach. In this paper, we present the algorithms and models for automatically deriving process models from versioning information, which we call *incremental workflow mining*. Our approach helps semi-automatically achieving the third level of the Capability Maturity Model (CMM) [Hum89] once the second level is reached.

An initial step for deriving the software process model is discovering the activities (a unit of work carried out by employees). Here, the essential part of an activity are the input and output documents and the users executing it – a document-oriented understanding of an activity. We call the algorithm for discovering activities *activity mining*. After detecting the sequence and parallel relations among activities, the *merging* algorithm discovers the structure of the process model in form of a Petri net. This model can be analyzed and verified using the wide range of Petri techniques; later it can be shown to the user in an understandable format. As soon as new audit information is available, the set of activities and the process model are refined and shown to the user again. This allows us to have an up-to-date model all the time. Consequently, we call our approach *incremental*.

The basic ideas of our approach were presented in our paper [KRS05b]; here, we do not deal with such steps as model analysis and model transformation to the user-oriented format. Rather, we sketch the idea of the *mining* and *merging algorithms*. For a detailed presentation of these algorithms we refer to an extended version of this paper [KRS05a].

2 Related Work

The research in the area of software *process mining* started with new approaches to the grammar inference problem for *event logs* [CW98]. The other work from the software domain are in the area of mining from software repositories [MSR05]; like in our approach, they use SCM systems and especially CVS as a source of input information, but for measuring the project activity, detecting and predicting changes in code, advising newcomers to an open-source project and detecting the social dependencies between the developers.

The first application of "process mining" to the *workflow domain* was presented by Agrawal in 1998 [AGL98]. This approach models business processes as annotated activity graphs and is restricted to sequential patterns. The approach of Herbst and Karagiannis [HK99] uses machine learning techniques for acquisition and adaptation of workflow models. The foundational approach to workflow mining was presented by van der Aalst et al. [WvdA01]. Within this approach, formal causality relations between events in logs and the α-mining algorithm for discovering workflow models with its improvements are presented. In addition to the software process and business process domains, the research concerning discovering the sequential patterns in the area of *data mining* is important here [AS95].

In comparison to the classical approaches, we do not have logs of activities and, so, must discover the activities first. We make use of our document-oriented view on the activities, i.e. the process is derived from the inputs and outputs of the activities. We suggest coming up with the model early and refining it when additional information is available.

3 Versioning Log

Next, we discuss the input information for our *incremental workflow mining* approach. Software Configuration Management (SCM) systems (e.g. CVS, SourceSafe, ClearCase) and Product Data Management (PDM) systems (e.g. Metaphase, Teamcenter) provide some information on the on-going work. This information contains data about *checkouts* and *checkins (commits)* of documents. In this paper, we deal only with checkin information since not all SCM systems enforce or support checkouts; checkins to the system are more accurate.

An example of a *versioning log* is shown in Table 1. We use a log format that is independent from particular SCM systems, but includes the typically available information.

Table 1: Versioning Log

Document	Date	Author	Comment
design	01.01.05 14:30	de	status: initial
code	01.01.05 15:00	de	status: generated
review	05.01.05 10:00	se	status: pending
testres	07.01.05 11:00	qa	status: initial, type: manual
design	01.02.05 11:00	de	status: initial
code	15.02.05 17:00	dev	status: generated
testres	20.02.05 09:00	dev	status: initial, type: manual
review	28.02.05 18:45	se	status: pending
design	01.03.05 11:00	de	status: initial
review	15.03.05 17:00	se	status: pending
code	20.03.05 09:00	dev	status: generated
testres	28.03.05 18:45	dev	status: initial, type: manual

A versioning log consists of *records* (rows). A record is produced for each checkin of a document during different executions of different software processes; a record contains information on the document name, the time, the author and the comment. In the log of Table 1, we have one process, let's call it "Design Change"; during this process, some software module is designed, code is generated and tested and the design is reviewed. There are three executions of this process: the first four records belong to the first execution, the second four – to the second execution, and the third four – to the third one. The set of records that belong to the same execution of one process is called an *execution log*.

For the algorithms presented in this paper, we make the following *assumptions* about the versioning log: First, different execution logs have the same naming conventions. E.g. the document containing the design is called "design" in all the execution logs. Second, the document name and the comment together are the unique identifier of the record. So, there is no documents with the same name and the same comment within one execution log. Third, there are no two records with the same timestamp in the same execution log. For the rest of the paper, we assume that we have only one process and know which records belong to which execution log; we can ignore the comments and deal only with document names; the timestamps are mapped to natural numbers, representing their order.

4 Activity Mining

Next, we present the algorithm for discovering the activities. As mentioned earlier, an *activity* is a logical unit of work defined by the sets of its input and output documents and by the set of resources allowed to execute it.

The *activity mining algorithm* goes through the logs looking for the output of the activities. Every document committed to the system is the output of some activity. For the log given in Table 1, the document "code" is produced by some activity (e.g. programming). For each activity, we can also get the resource who committed the document. E.g. the document “code” was committed by “de” in the first execution log and by “dev” in the second and the third one.

The important problem here is finding the input of each activity, since this information is not given in the logs explicitly. For now, we assume that all documents that were always committed to the system earlier than the output of the activity belong to its input. E.g. for the activity producing “code”, the document “design” was committed earlier in all the execution logs, thus it is the input of the activity; the document “review” was also committed earlier, but only in the third log, thus it does not belong to the input set. Thus, the activity producing “code” is the following tuple $(\{design\}, \{code\}, \{dev, de\})$. The first set is the input of the activity, the second is the output, and the third is the set of resources. For details on the *activity mining* algorithm, we refer to the extended paper [KRS05a].

5 Merging

The merging algorithm generates the software process model from the set of activities. Since the set of activities obtained by activity mining contains the essence of the versioning log needed for deriving the process model, we do not deal with the log anymore.

First, we define three relations on the set of activities: sequence, parallel split, and parallel join. The *sequence* relation contains the pairs of activities that follow each other, i.e. the input of the second activity must contain the input and the output of the first one. The *parallel split* relation contains the activities which have the same input, but produce different output; these activities are concurrent. The *parallel join* relation contains activities that are followed by an activity that needs the input and the output of both activities. The merging algorithm generates the control, the informational and the organizational aspects of the process model as a Petri net, see Fig. 1. The exact algorithm is precisely defined in our technical report [KRS05a].

The *control aspect algorithm* identifies the order on the activities. For each activity, it generates a transition. E.g., for the activity producing “code”, it generates transition t_{code}. For each activity, which is not followed by parallel activities, it also generates one output place and an arc between them, e.g. for the “code” activity, it generate p_{code} and (t_{code}, p_{code}). If an activity is followed by parallel activities, we introduce a *parallel split pattern*, which consists of a set of places – one for each parallel branch; a transition of the activity is connected to all the places. E.g. the activities producing “code” and “review” are in parallel split relation; therefore, we connect transition of the preceding activity, which produces “design”, to these places. Then, the places are connected to the corresponding transitions. Next, we connect all activities that are in *sequence relation*, except for the ones in parallel split relation. For example, we connect the place p_{code} to transition $t_{testres}$. The last step is connecting the *parallel join pattern*. Here, we connect the output places of activities that belong to the parallel join relation with the transition of activities that needs the documents produced by all of these activities as input. For example, an activity which produces the “end” document needs the documents of its predecessors for input.

For the *informational aspect* of the process model, we create one place for each document.

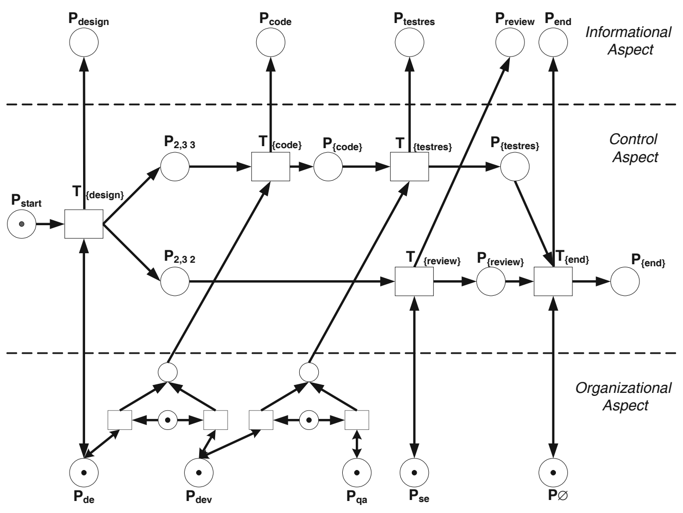

Figure 1: The Process Model

Since each activity produces a document, we can connect the corresponding transitions to these document places. For the *organizational aspect* of the process model, we create one place for each existing user and put a token to each of these places to show that the user is available. If an activity can be executed only by one user, we create an edge from the user place to the corresponding transition and back. If an activity can be executed by one of several users, we introduce a special exclusive choice construct between these users.

6 Conclusion and Future Work

The activity mining and merging algorithms were implemented in Prolog as a part of the incremental workflow mining research prototype. The declarative semantics of Prolog was especially valuable for implementing these algorithms. A special clause generates the resulting Petri net, which can be already shown to the user.

In our approach, we do mining from different perspectives, we start with deriving the information from the logs of the SCM system. Our approach has the following benefits: First, it does not need information on the existing activities; rather, we can mine this information from the versioning logs. Second, after deriving the set of activities from the versioning log, we do not deal with logs anymore. Third, our approach can be used incrementally, since we do not have to deal with all the logs after getting the new execution log again. Finally, our algorithms produce a multi-perspective process model, which reflects the real process carried out in a company. Consequently, the approach helps gradually improving the management of the software processes of a company along with incrementally improving the process models.

In this paper, we presented algorithms which work under rather restrictive assumptions on the versioning log. Future extensions will relax these assumptions: several documents can be committed at the same time, there can be several commits of the same document within one execution log. Additionally, we must identify the types of documents, the roles of users and we must generate meaningful names for the activities. This can be achieved by exploiting checkout information of SCM systems or information of the additional tools and by interacting with the user. For discovering not only software processes, but system engineering processes in general, we shall deal with similar audit information that can be obtained from Product Data Management (PDM) systems or other types of configuration and change management systems.

Altogether, our approach supports mining process models from ad-hoc executions without having predefined information on them. This way, our approach supports increasing the repeatability of the processes, in terms of workflow terminology the ad-hoc processes become administrative. In terms of the CMM, this means automatic support for increasing the maturity level of some enterprise from repeatable to defined.

References

[AGL98] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Proceedings of the 6th International Conference on Extending Database Technology*, pp. 469–483. Springer, 1998.

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. In P. S. Yu and A. S. P. Chen, eds., *Eleventh International Conference on Data Engineering*, pp. 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[CW98] J. E. Cook and A. L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, 1998.

[HK99] J. Herbst and D. Karagiannis. An Inductive approach to the Acquisition and Adaptation of Workflow Models. citeseer.ist.psu.edu/herbst99inductive.html, 1999.

[Hum89] W. S. Humphrey. *Managing the software process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[KRS05a] E. Kindler, V. Rubin, and W. Schäfer. Activity Mining and Discovering Software Process Models. Technical Report tr-ri-05-265, University of Paderborn, http://www.upb.de/cs/ag-schaefer/Personen/Aktuell/Rubin/TR/tr-ri-05-265.pdf, 2005.

[KRS05b] E. Kindler, V. Rubin, and W. Schäfer. Incremental Workflow Mining based on Document Versioning Information. In M. Li, B. Boehm, and L. J. Osterweil, editors, *Proc. of the Software Process Workshop 2005, Beijing, China*, LNCS 3840. Springer, May 2005.

[MSR05] *MSR 2005 International Workshop on Mining Software Repositories*, New York, NY, USA, 2005. ACM Press.

[WvdA01] A.J.M.M. Weijters and W.M.P. van der Aalst. Process mining: discovering workflow models from event-based data. In *Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001)*, pp. 283–290, 2001.