Process Mining Framework for Software Processes

Vladimir Rubin[1,2], Christian W. Günther[1], Wil M.P. van der Aalst[1], Ekkart Kindler[2], Boudewijn F. van Dongen[1], and Wilhelm Schäfer[2]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
{c.w.gunther,w.m.p.v.d.aalst,b.f.v.dongen}@tue.nl
[2] University of Paderborn, Paderborn, Germany
{vroubine,kindler,wilhelm}@uni-paderborn.de

Abstract. Software development processes are often not explicitly modelled and sometimes even chaotic. In order to keep track of the involved documents and files, engineers use Software Configuration Management (SCM) systems. Along the way, those systems collect and store information on the software process itself. Thus, SCM information can be used for constructing explicit process models, which is called *software process mining*. In this paper we show that (1) a Process Mining Framework can be used for obtaining software process models as well as for analysing and optimising them; (2) an algorithmic approach, which arose from our research on software processes, is integrated in the framework.

1 Introduction

Software and information systems are still becoming more and more complex. One of the distinguishing features of any engineering effort is the fact that process engineers create, change, update and revise all kinds of documents and files. In order to cope with the vast amount of data, documents, and files, engineers use Product Data Management (PDM) systems or Software Configuration Management (SCM) systems such as CVS or Subversion. In addition to maintaining the engineer's documents, these systems collect and store information on the process: Who created, accessed, or changed which documents?, When was a particular task completed?, etc.

The engineering processes themselves, however, are often not well-documented and sometimes even chaotic: engineering processes tend to be far less structured than production processes. In order to help engineers to identify, to better understand, to analyse, to optimise, and to execute their processes, the process data stored in the SCM systems can be used for extracting the underlying engineering processes and for automatically constructing one or more explicit *process models*. We call this *software process mining*.

Process models and software process models cover different aspects. Here, we consider the main aspects only: the *control aspect* captures the order in which tasks are executed (i. e. the control-flow), the *information aspect* captures the data, documents, and information needed and produced by a task,

and the *organisation aspect* captures which persons in which role execute a task. To mine different aspects of software development processes – sometimes called *multi-perspective mining* – we need different algorithms. In order to make all these algorithms available under a single user interface, we use the ProM framework [1]. ProM provides a variety of algorithms and supports process mining in the broadest sense. It can be used to discover processes, identify bottle-necks, analyse social networks, verify business rules, etc. Moreover, ProM provides interfaces to extract information from different sources including SCM systems such as CVS and Subversion.

The focus of this paper is on providing an overview of the application of process mining to software processes. Although we do not focus on the algorithms, we discuss one process mining algorithm, which was specifically developed for software processes and integrated in ProM. Moreover, we discuss in which other ways ProM can help software engineers in dealing with their processes.

2 Related Work

The capabilities of using software repositories for deriving information about the software projects are being researched in the domain of *mining software repositories* [2]. Like in our approach, SCM systems are used as sources of information. They are used for measuring the project activity and the amount of produced failures, for detecting and predicting changes in the code, for providing guidelines to newcomers to an open-source project, and for detecting the social dependencies between the developers. In this area, SCMs are mostly used for detecting dependencies on the code level, whereas we make an effort at building process models and analysing them. Researchers and practitioners recognize the benefits of *software process modelling* with the aid of software repositories [3, 4]. Nowadays, process improvement should be ruled by what was actually done during the software development process and not by what is simply said about it. The researchers from this domain examine *bug reports* for detecting defect lifecycles, *e-mails and SCMs* for analysing the requirement engineering processes and coordination processes between developers, their productivity and participation, etc. Although this research direction deals with software processes and their models, there is still a lack of algorithms for producing formal models.

Since the mid-nineties several groups have been working on techniques for process mining, i.e., discovering process models based on observed events. In [5], an overview is given of the early work in this domain. The idea to apply process mining in the context of workflow management systems was introduced in [6]. However, we argue that the first papers really addressing the problem of process mining appeared around 1995, when Cook et al. [7, 8] started to analyse recorded behaviour of processes in the context of software engineering, acknowledging the fact that information was not complete and that a model was to be discovered that reproduces *at least* the log under consideration, but it may allow for more behaviour. More information on recent process mining research can be found at *http://www.processmining.org*.

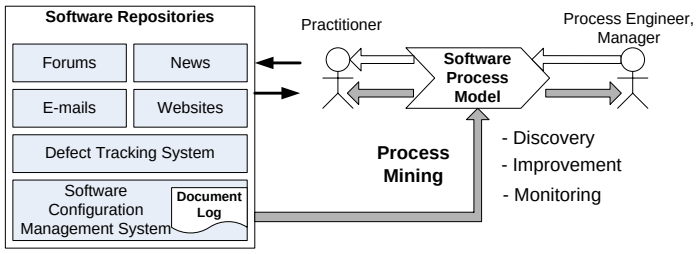

Fig. 1. Process-centered Software Engineering and Process Mining

3 Process Mining for Software Engineering Environments

In this section, we first explain the traditional process-centered software engineering environments (PSEE). Then, we present the ideas of the *incremental workflow mining* approach.

3.1 Incremental Workflow Mining Approach

Figure 1 gives an overview of the architecture of a traditional PSEE and represents how our *incremental workflow mining* approach is integrated to this architecture: The environment consists of software repositories (SCM system, defect tracking system, etc...). The *software product* and the interaction among *practitioners* are supported and maintained by the repositories. In the traditional schema, the *Process Engineer* (project manager or department) designs the process model using his experience and existing approaches, like V-model, RUP, etc. Then, the model is instantiated and practitioners follow it during the product life cycle, indicated by the white arrows in Fig. 1. There are the following problems with this schema: The designed process model does not necessarily reflect the actual way of work in the company, human possibilities in detecting discrepancies between the process model and the actual process are limited, practitioners are not involved in the design of the process model.

The main ideas of the *incremental workflow mining* approach were described already in our previous work [9, 10]. In this approach we go the other direction, it is shown with gray arrows in Fig. 1: We take the *audit trail information* (document log) of the SCM system, which corresponds to the process instances (particular executions of the process) and, using our *process mining algorithms*, derive the process model from it. Then, the process model can be analysed, verified and shown to the process engineer; he decides which changes should be introduced to the process to optimise and to manage it in a better way. Actually, the mining approach can be used not only for *discovery*, but also for *monitoring* and *improving* real software processes using the data from software repositories in general and SCM systems in particular.

In software engineering environments, it is usually difficult to introduce a *Process Management System* (PMS) directly from scratch. Using our approach

Table 1. Document Log

Document	Date	Author
project1/models/design.mdl	01.01.05 14:30	designer
project1/src/Code.java	01.01.05 15:00	developer
project1/tests/testPlan.xml	05.01.05 10:00	qaengineer
project1/docs/review.pdf	07.01.05 11:00	manager
project2/models/design.mdl	01.02.05 11:00	designer
project2/tests/testPlan.xml	15.02.05 17:00	qaengineer
project2/src/NewCode.java	20.02.05 09:00	developer
project2/docs/review.pdf	28.02.05 18:45	designer
project3/models/design.mdl	01.03.05 11:00	designer
project3/models/verification.xml	15.03.05 17:00	qaengineer
project3/src/GenCode.java	20.03.05 09:00	designer
project3/review/Areview.pdf	28.03.05 18:45	manager

Table 2. Filtered Log

Document
DES
CODE
TEST
REV
DES
TEST
CODE
REV
DES
VER
CODE
REV

in a *batch mode*, we gather the existing logs of several process instances and automatically generate a model from them. Our approach works also *incrementally*, i.e. as soon as new data is added to the repositories, we refine the overall process model. Following this approach, the role of the PMS changes over time: at the beginning, it is utilized only for storing the newly discovered models; after model improvements, the system can start advising the users and controlling their work in the company. We call this *gradual process support*.

3.2 Input Information

In this section, we focus on the logs of SCM systems and make our experiments with them, but the approach and the algorithms are more general: They also deal with the information derived from other software repositories.

In Table 1, we present an example of the audit trail information from an SCM system. SCM systems record the *events* corresponding to the *commits of documents*. A sequence of these events constitutes a *document log*: It contains the names of the committed documents, timestamps, and author names. Document logs with similar structure can be derived from all kinds of SCM systems, such as *CVS*, *Subversion*, *SourceSafe*, *Clear Case* and others. When we analyse the document logs, we have to identify the cases (process instances), identify the document types, abstract from the details of the log, and ignore unnecessary information. For many software projects, a *case* corresponds to a subproject or a plug-in development, in our example it corresponds to a project development (cases are separated with double lines in the tables). We detect the *documents' types* by identifying similarities of their paths and names, see Sect. 4.1 for details. The same technique is used for abstracting from the log details and for ignoring noise, i.e. ignoring exceptional or infrequent commits. However, the latter issues are also resolved on the algorithm level, see Sect. 4.2.

4 Process Mining Algorithms and Tool Support

In this section, we present the algorithms for multi-perspective software process mining. In the area of process mining, there are different algorithmic approaches, which derive the control-flow, the organization and the information models from the *event logs*. The events in these logs correspond to process *activities* produced by some PMS. In our application area, we have information about the *commits of documents* which occur in SCM systems, but generally can also occur in other systems, like PDM. All the presented algorithms are integrated as plug-ins to the *ProM* tool [1], which is described at the end of this section.

4.1 Abstraction on the Log Level

The document logs often contain either too many details or very specific document names and paths, which are not relevant for the process mining algorithms. Thus, we need a technique to abstract from the concrete names and paths or even to ignore some paths. We call this *abstraction on the log level*. The ProM tool contains a set of *filters*, which help us solving this problem.

Here, we use the remap filter, which maps the names of documents from the log to abstract names. Regular expressions specify the paths that should be mapped to abstract names. For example, if the path contains "/models/", the filename contains "design" and has extension ".mdl", then it should be mapped to "DES". Table 2 shows the result of this filter applied to the log of Table 1.

4.2 Control-flow Mining

In this section, we describe the control-flow mining algorithms. When dealing with the control-flow, the log can be represented as a set of sequences of documents (sequences are also called cases, traces or execution logs), see Table 2.

Generation and Synthesis Approach The approach presented in this section is a *two-step approach*: Step 1 takes a document log and generates a transition system (TS) from it; Step 2 synthesises a Petri Net (PN) from the transition system. The algorithmic details of the approach are discussed in [11]. One of the main advantages of the approach is the capability to *construct* transition systems and, then, to apply different *modification strategies* depending on the desired degree of generalization; we call this "clever" transition system generation or *abstraction on the model level*. Despite the fact that transition systems are a good specification technique for making experiments, they are usually huge, since they encode such constructs as concurrency or conflict in a sequential way. Thus, the algorithms developed within such a well-known area of Petri net theory as *Petri net synthesis and theory of regions* [12] are used for transforming transition systems to Petri nets, which are more compact.

The transition system shown in Fig. 2(a) with the solid arrows is *constructed* from the log given in Table 2. In this example, a *state* is defined as a *set* of

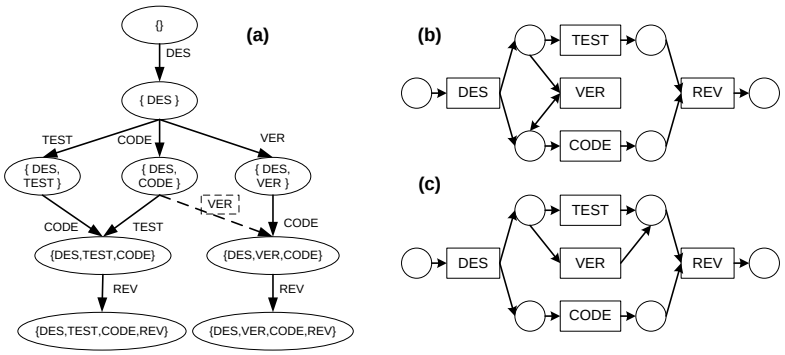

Fig. 2. Generated and Synthesis Approach: (a) Transition Systems (b),(c) Petri Nets

documents representing the complete history of a case at a point of time. For example, for the first case, there are such states as $\{\}$, $\{DES\}$, etc. There are transitions between all the subsequent pairs of states, transitions are labelled with the names of produced documents. Using the Petri net synthesis algorithms, we generate a Petri net from the given TS, see Fig. 2(b). Events of the TS correspond to the transitions of the PN. This Petri net has the same behaviour as the TS; the concurrency of events $TEST$ and $CODE$, which is modeled sequentially in the TS, is specified more compact in the PN.

But we can also *modify* the constructed TS using some strategy. For example, the "Extend Strategy" adds transitions between two states, which were created from different traces but which can be subsequent because there is a single document which can be produced to reach one state from the other. As a result, we add one transition VER from state $\{DES, CODE\}$ to state $\{DES, VER, CODE\}$, it is shown with the dashed arrow in Fig. 2(a). A Petri net corresponding to this TS is shown in Fig. 2(c). This Petri net is more general than the first one; it allows an additional trace, namely $\langle DES, CODE, VER, REV\rangle$.

The first ideas of the generation and synthesis approach were presented in our previous paper [13], then the algorithms were significantly improved and successfully implemented in the context of ProM; the tool Petrify [14] is used in the synthesis phase. This approach overcomes many limitations of the traditional process mining approaches; for example, it can deal with *complicated process constructs*, *overfitting* (generated model allows only for the exact behaviour seen in the log) and *underfitting* (model overgeneralises the things seen in the log). However, by now, this approach can hardly deal with *noise* (incorrectly logged events and exceptions), since we do not consider the frequencies of cases in the log; so, the other approaches that treat this problem, are presented in the next Section.

Other Approaches for Control Flow Mining In the process mining domain a number of algorithms for control flow mining have been developed, which

have different characteristics from the previously introduced approach; all these algorithms can be also applied for mining the software processes.

The *Alpha algorithm* [15] can also derive a Petri net model from an event log, however it is based on analysing the immediate successor relation between event types, i.e. documents. Another algorithm, the *Multi-phase approach* [16], creates Event-driven Process Chain (EPC) models from a log, while it first generates a model for each process instance and later aggregates these to a global model. Both the Alpha and the Multi-phase algorithms share the generation and synthesis approach's *precision*, i.e. the generated model accurately reflects all ordering relations discovered in the log.

While sophisticated filtering of logs can remove noise partially, there are also process mining algorithms which are designed to be more robust in the presence of noise. The *Heuristics Miner* [17] employs heuristics which, based on the frequency of discovered ordering relations, attempts to discard exceptional behaviour. Another approach in this direction is the *Genetic Miner* [18]. It uses genetic algorithms to develop the process model in an evolutionary manner, which enables it to also discover e.g. long-term dependencies within a process.

4.3 Mining other perspectives

Our generation and synthesis approach deals with the control flow, which is only one *perspective* addressed in process mining. Such information as the timestamp of an event or its originator (the person having triggered its occurrence) can be used to derive high-level information about the process also in other perspectives.

Resource Perspective. The resource perspective looks at the set of people involved in the process, and their relationships. The *Social Network Miner* [19] for example can generate the *social network* of the organization, which may highlight different relationships between the persons involved in the process, such as *handover of work*, *subcontracting* and others. The *Organizational Miner* also addresses the resource perspective, attempting to cluster resources which perform similar tasks into *roles*. This functionality can be very beneficial in a software development process, both for verification and analysis of the organizational structure. Mismatches between discovered and assigned roles can pinpoint deficiencies in either the process definition or the organization itself.

Performance Perspective. Mining algorithms addressing the *performance* perspective mainly make use of the timestamp attribute of events. From the combination of a (mined or predefined) process model and a timed event log, they can give detailed information about performance deficiencies, and their location in the process model. If some project phase is identified as the point in the process where most time is spent, we could assign more staff to this task.

Information Perspective. The *Activity Miner* [20] can derive high-level activities from a log by clustering similar sets of low-level events that are found to occur together frequently. These high-level clusters, or patterns, are helpful for unveiling hidden dependencies between documents, or for a re-structuring of the document repository layout.

4.4 Process Analysis and Verification

Process mining is a tremendously helpful tool for managers and system administrators, who want to get an overview of how the process is executed, and for *monitoring* progress. However, in many situations it is interesting whether execution is *correct*. To answer this question, there exists a set of *analysis and verification methods* in the process mining domain. One of these techniques is *Conformance Checking* [21], which takes a log and a process model, e.g. a Petri net, as input. The goal is to analyse the extent to which the process execution corresponds to the given process model. Also, conformance checking can point out the parts of the process where the log does not comply.

Another technique is *LTL Checking* [22], which analyses the log for compliance with specific constraints, where the latter are specified by means of linear-temporal logic (LTL) formulas. In contrast to conformance checking, LTL checking does not assume the existence of a fully defined development process. Therefore, it can be used to successively introduce, and check for, corporate guidelines or best development practices.

The ProM framework also features techniques for process model analysis and verification in the absence of a log. Advanced process model analysers, such as *Woflan*, can check e.g. a Petri net model for *deadlocks* (i.e., potential situations in which execution will be stuck), or verify that all process executions complete properly with no enabled tasks left behind. Process designers find these automated tools valuable for ensuring that a defined development process will not run into problems which are hard to resolve later on.

4.5 ProM and ProMimport Tools

The ideas presented in this paper have been implemented in the context of *ProM*. ProM serves as a testbed for our process mining research [1] and can be downloaded from www.processmining.org. Starting point for ProM is the MXML format. This is a vendor-independent format to store event logs. One MXML file can store information about multiple processes. Per process, events related to particular process instances (cases) are stored. Each event refers to an activity. In the context of this paper, documents are mapped onto activities. Events can also have additional information such as the transaction type (start, complete, etc.), the author, timestamps, and arbitrary data (attribute-value pairs).

The *ProMImport Framework* allows developers to quickly implement plug-ins that can be used to extract information from a variety of systems and convert it into the MXML format (cf. promimport.sourceforge.net). There are standard import plug-ins for a wide variety of systems, e.g., workflow management systems like Staffware, case handling systems like FLOWer, ERP components like PeopleSoft Financials, simulation tools like ARIS and CPN Tools, middleware systems like WebSphere, BI tools like ARIS PPM, etc. Moreover, it has been used to develop many organization/system-specific conversions (e.g., hospitals, banks, governments, etc.). The ProMImport Framework can also be used to extract event logs from such systems as Subversion and CVS.

Once the logs are converted to MXML, ProM can be used to extract a variety of models from these logs. ProM provides an environment to easily add plug-ins that implement a specific mining approach. The most interesting plug-ins in the context of this paper are the mining plug-ins. In addition to that, there are four other types of plug-ins: *Export plug-ins* implement some "save as" functionality for some objects (such as graphs). For example, there are plug-ins to save EPCs, Petri nets, spreadsheets, etc. *Import plug-ins* implement an "open" functionality for exported objects, e.g., load instance-EPCs from ARIS PPM. *Analysis plug-ins* typically implement some property analysis on some mining result. For example, for Petri nets, there is a plug-in which constructs place invariants, transition invariants, and a coverability graph. *Conversion plug-ins* implement conversions between different data formats, e.g., from EPCs to Petri nets and from Petri nets to YAWL and BPEL. Altogether, there are *140* plug-ins for ProM.

5 Evaluation and Applications

In order to evaluate our approach, we have chosen the *ArgoUML* project, which is an open-source UML modeling tool maintained by the *Subversion* SCM system. Since this data is freely available, it makes an excellent test case for us. ArgoUML has different subprojects with the same *file organization*; we have chosen five subprojects which implement the ArgoUML support for five different programming languages. We will use these five process instances to derive a formal model of the control-flow, to analyse the organization structure and the performance of the process, and to do some analysis and verification.

First, using the *svn log* utility provided by Subversion, we generated logs for all the five subprojects and imported them to ProM. This log consisted of about 400 commit events. The log contains project specific paths and different commits, which are not relevant for the software process. Using the *remap filter*, we replaced project specific paths with abstract names. Following the ArgoUML conventions, all the committed documents (files) containing "/src/" in their paths and have ".java" as an extension were mapped to "SRC", all the "readme.*" files – to "README", all the files in "/tests/" – to "TESTS", the files in "/www/" – to "WWW", "build.bat" – to "BUILDER" and all the files, which names start with "." – to "CONFIG"; the other commits were ignored.

After executing the algorithms of the *generation and synthesis* approach, we obtained the Petri net shown in Fig. 3. Here, for the sake of readability, we show a simplified Petri net without loops – which was obtained by applying the "Kill Loops" modification strategy to the transition system and synthesizing a Petri net from there. Thus, the Petri net focuses on the start events, i.e. when source code development was started, when testing was started. People use to start with building web sites or editing readme files and builders, then they write code and then, they test it, sometimes builder file is changed after writing code.

The Petri net model of the development process can now be used for enhanced analysis within the ProM framework. Figure 4 shows the result of a *performance analysis* based on the mined model and the log. The states, i.e. places, have

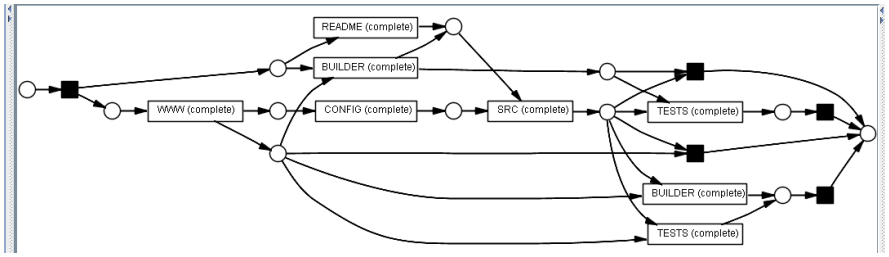

Fig. 3. Petri Net for the ArgoUML Project

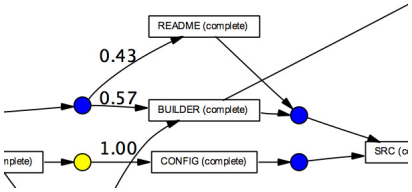

Fig. 4. Performance Analysis

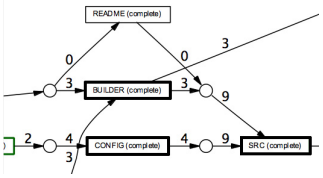

Fig. 5. Conformance Analysis

been colored according to the time which is spent in them while executing the process. Also, multiple arcs originating from the same place (i.e., choices) have been annotated with the respective probability of that choice.

Further, a *conformance analysis* can be performed using the Petri net model and the associated log. Figure 5 shows the *path coverage* analysis of the conformance checker. All activities that have been executed in a specific case (in our example we chose the C++ language support) are decorated with a bold border, and arcs are annotated with the frequency they have been followed in that case. This example shows, that the C++ team did not create a README file.

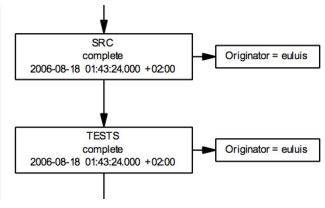

Fig. 6. LTL Analysis

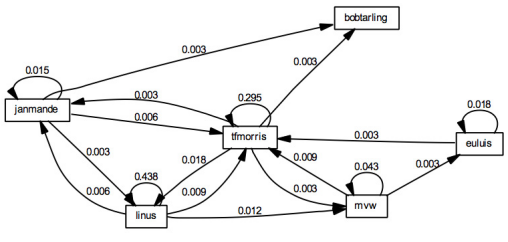

Fig. 7. Social Network

One known software engineering concept is the "four eyes principle", e.g. developers working on the source code should not write tests as well. Figure 6 shows the result of checking a corresponding *LTL* formula on the ArgoUML log. In the C++ support case, which is shown in Fig. 6, both source code and tests have been submitted by the developer "euluis", thereby violating this principle.

For determining the *social network* of a development process, it is preferable to use the original log, i.e. before it has been abstracted like explained in Section 4.1. The reason for that is, that it is also interesting when people collaborate within a certain part of the project (e.g. writing source code), while one wants to abstract from these activities on the control flow level. Figure 7 illustrates the hand-over of work between ArgoUML developers. It shows that some developers are involved only in specific phases of the project (e.g. "bobtarling" appears to work only at the end of projects), while others (e.g. "tfmorris") have a more central and connected position, meaning they perform tasks all over the process. Based on the nature of the project one may prefer different collaboration patterns, which can be checked conveniently in a mined social network like this.

6 Conclusion

In this paper, we have discussed some new algorithms for mining software and systems engineering processes from the information that is available in Software Configuration Management Systems. These algorithms are included in the ProM framework, which has interfaces to a variety of document management systems. Therefore, ProM is now an effective tool for software process mining.

For evaluation purposes, we have mined the software processes of a real project: ArgoUML. This shows that we can obtain the process models for realistic software projects. Moreover, we have shown that ProM could be used for analysing and verifying some properties of these processes.

Acknowledgements This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Dutch Ministry of Economic Affairs.

References

1. van Dongen, B., Medeiros, A., Verbeek, H., Weijters, A., van der Aalst, W.: The ProM framework: A New Era in Process Mining Tool Support. In Ciardo, G., Darondeau, P., eds.: Application and Theory of Petri Nets 2005. Volume 3536. (2005) 444–454
2. MSR 2005 International Workshop on Mining Software Repositories. In: ICSE '05: Proc. of the 27th International Conference on Software Engineering, New York, NY, USA, ACM Press (2005)
3. Sandusky, R.J., Gasser, L., Ripoche, G.: Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community. In: MSR 2004: International Workshop on Mining Software Repositories. (2004)
4. Iannacci, F.: Coordination Processes in Open Source Software Development: The Linux Case Study. http://opensource.mit.edu/papers/iannacci3.pdf (2005)

5. van der Aalst, W., van Dongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, A.: Workflow Mining: A Survey of Issues and Approaches. Data and Knowledge Engineering **47** (2003) 237–267
6. Agrawal, R., Gunopulos, D., Leymann, F.: Mining Process Models from Workflow Logs. In: Sixth International Conference on Extending Database Technology. (1998) 469–483
7. Cook, J., Wolf, A.: Discovering Models of Software Processes from Event-Based Data. ACM Trans. on Software Engineering and Methodology **7** (1998) 215–249
8. Cook, J., Du, Z., Liu, C., Wolf, A.: Discovering models of behavior for concurrent workflows. Computers in Industry **53** (2004) 297–319
9. Kindler, E., Rubin, V., Schäfer, W.: Incremental Workflow mining based on Document Versioning Information. In Li, M., Boehm, B., Osterweil, L.J., eds.: Proc. of the Software Process Workshop 2005, Beijing, China. Volume 3840., Springer (2005) 287–301
10. Kindler, E., Rubin, V., Schäfer, W.: Activity mining for discovering software process models. In Biel, B., Book, M., Gruhn, V., eds.: Proc. of the Software Engineering 2006 Conference, Leipzig, Germany. Volume P-79 of LNI., Gesellschaft für Informatik (2006) 175–180
11. van der Aalst, W., Rubin, V., van Dongen, B., Kindler, E., Günther, C.: Process Mining: A Two-Step Approach using Transition Systems and Regions. BPM Center Report BPM-06-30, BPM Center, BPMcenter.org (2006)
12. Cortadella, J., Kishinevsky, M., Lavagno, L., Yakovlev, A.: Deriving Petri nets from finite transition systems. IEEE Trans. on Computers **47** (1998) 859–882
13. Kindler, E., Rubin, V., Schäfer, W.: Process Mining and Petri Net Synthesis. In Eder, J., Dustdar, S., eds.: BPM 2006 Workshops. Volume 4103., Springer (2006)
14. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. IEICE Trans. on Information and Systems **E80-D** (1997) 315–325
15. van der Aalst, W., Weijters, A., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Trans. on Knowledge and Data Engineering **16** (2004) 1128–1142
16. van Dongen, B., van der Aalst, W.: Multi-Phase Process Mining: Building Instance Graphs. In Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T., eds.: International Conference on Conceptual Modeling (ER 2004). Volume 3288. (2004) 362–376
17. Weijters, A., van der Aalst, W.: Rediscovering Workflow Models from Event-Based Data using Little Thumb. Integrated Computer-Aided Engineering **10** (2003) 151–162
18. van der Aalst, W., Medeiros, A., Weijters, A.: Genetic Process Mining. In Ciardo, G., Darondeau, P., eds.: Applications and Theory of Petri Nets 2005. Volume 3536. (2005) 48–69
19. van der Aalst, W., Reijers, H., Song, M.: Discovering Social Networks from Event Logs. Computer Supported Cooperative work **14** (2005) 549–593
20. Günther, C., van der Aalst, W.: Mining Activity Clusters from Low-level Event Logs. BETA Working Paper Series, WP 165, EUT, Eindhoven (2006)
21. Rozinat, A., van der Aalst, W.: Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In Bussler et al., C., ed.: BPM 2005 Workshops. Volume 3812. (2006) 163–176
22. van der Aalst, W., Beer, H., Dongen, B.: Process Mining and Verification of Properties: An Approach based on Temporal Logic. BETA Working Paper Series, WP 136, Eindhoven University of Technology, Eindhoven (2005)