Process Mining Framework for Software Processes

V. Rubin[1,2], C.W. Günther[1], Wil M.P. van der Aalst[1], E. Kindler[2], B.F.van Dongen[1], and W. 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 systems. Along the way, those systems collect and store information on the software development process itself. In this paper, we show how this information can be used for constructing explicit process models, which is called process mining; and we show how the Process Mining Framework ProM can help engineers in obtaining a process model and in analysing, optimising and better understanding their software processes.

1 Introduction

Software and information systems are still becoming more and more complex. This is a result of the increased functionality and a much higher integration and interaction of different kinds of software and systems. Therefore, the process of software and system's engineering is an important challenge. In fact, the engineering of software has become more complicated than the classical engineering of physical products. 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 different kinds of *Document Management Systems* such as Product Data Management (PDM) systems, repositories, and Software Configuration Management (SCM) systems (e.g., CVS and Subversion). In addition to the requirements, design and implementation documents, these systems collect and store information on the actual engineering 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. Despite the use of document management systems, 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 engineering processes, the process data stored in the Software Configuration Management, can be used for making the underlying

engineering processes explicit in terms of one or more *process models*. We call this *software process mining*, i.e., by extracting information from systems such as CVS and Subversion and analyzing this information, new models are generated or existing models are confronted with reality.

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 mining different aspects of software development processes – sometimes called *multi-perspective mining* – we need to exploit different algorithms. Sometimes, we need even different algorithms for mining a model for a single aspect. In order to make all these algorithms available under a single user interface, we use the ProM framework [21] as a platform. ProM provides a wide variety of algorithms and supports process mining in the broadest sense. It can be used to discover processes, identify bottle-necks, analyze social networks, verify business rules, etc. Using ProMImport it is also possible 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 development processes. Although we do not focus on particular algorithms, we introduce a two-step approach to extracting Petri nets from event logs. In the first step a transition system is constructed and in the second step this transition system transformed into a Petri net. Moreover, we discuss in which other ways ProM can help software engineers in dealing with their processes.

The remainder of this paper is organized as follows. First, we present related work. Then, we discuss the application of process mining in software engineering environments. Then, we provide an overview of process mining approaches and the tool support offered by ProM. In Section 5 we show the application of process mining to the Subversion logs of the ArgoUML project where we analyzed five subprojects. Section 6 concludes the paper.

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* [1, 2]. Like in our approach, SCM systems such as CVS and Subversion are used as sources of input 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 [32, 40, 18]. In this area, SCMs are mostly used for detecting dependencies on the code level, whereas we make an effort at building process models and doing analysis on the modeling level.

Researchers and practitioners recognize already the benefits of *software process modelling* with the aid of software repositories [36, 26, 33, 37]. Formerly, process modelling and improvement was mainly based on what practitioners said about their process (interviews, questionnaires); nowadays, process improvement can be ruled by what was actually done during the software development process. 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 is dealing with software processes and process models, there is still a lack of algorithms for producing formal models.

In addition to the software process and business process domains, the research concerning discovering the sequential patterns treats similar problems in the area of *data mining*. The work of Agrawal and Srikant deals with discovering sequential patterns in the databases of customer transactions [10].

Since the mid-nineties several groups have been working on techniques for process mining, i.e., discovering process models based on observed events. In [6] 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 [9]. However, we argue that the first papers really addressing the problem of process mining appeared around 1995, when Cook et al. [13, 15, 11, 12, 14] started to analyze 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.

A complete overview of recent process mining research is beyond the scope of this paper. Therefore, we limit ourselves to a brief introduction to this topic and refer to [6, 7] and the http://www.processmining.org web page for a more complete overview.

3 Process Mining for Software Engineering Environments

In this section, we first explain a traditional software engineering environment schema inspired by the work in the area of process-centered software engineering environments (PSEE) and software processes [19, 23, 24, 34]. Then, we present the ideas of the *incremental workflow mining* approach and, finally, explicitly state the main problem treated in this paper.

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, such as *SCM* system, *defect tracking* system, *e-mail* and *websites*, *news* archives and discussion *forums*. The *software product* (models, documents, source code, etc...) and the interaction between *practitioners* (the users who develop this product) are

supported and maintained in the repositories. In the traditional schema, the *Process Engineer* (project manager or process engineering department) designs the process model using his experience and existing approaches, like V-model [22], RUP [27], etc. Then, the model is instantiated and practitioners follow it during the product life cycle, indicated by the white arrows in Fig. 1.

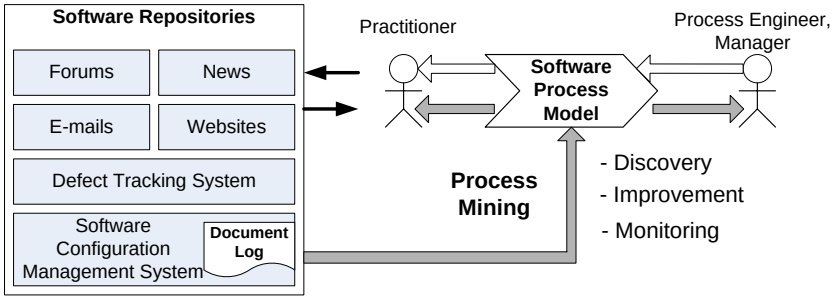

Fig. 1. Process-centered Software Engineering Environment: Traditional and Incremental Workflow Mining Schemes

There are the following problems with this schema: The designed process model is prescriptive, i.e. it 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, in spite of the fact that they are the best specialists in the parts of the process that they carry out.

The main ideas of the *incremental workflow mining* approach were described already in our previous work [28, 29]. 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 analyzed, verified and shown to the process engineer; he decides which changes should be introduced to the process to optimize 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. So, process mining is useful not only in a setting where there is *no explicit process model* and much *flexibility* is allowed (it is especially relevant for the software development processes), but also in a setting where the model exists already.

In the software engineering environments, it is usually difficult to introduce a *process management system* directly from scratch. The real process is very complicated and different people are working concurrently on different parts of the project. Using our approach in a *batch mode*, we gather the existing logs of

Table 1. Document Log

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

several process instances and automatically generate a model from them. Still, it is almost impossible to do it in one step without incrementally improving the knowledge of the process management system and the people's habit of relying on it. Thus, 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, when the process models are discovered, they must be inserted to the *Process Management System*, where they are maintained and executed. The role of the Process Management System changes over time: at the beginning, it is utilized only for storing the newly discovered models; after further refinements, when process models become more faithful, the system starts advising the users and controlling their work in the company. We call it *gradual process support*.

3.2 Input Information and Problem Statement

According to the description of the incremental workflow mining approach given in the previous section, the input information comes from software repositories. At present, 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* (documents are committed to the system by the users). The sequence of these events constitutes a *document log*. The document log contains the following information: names of the committed documents, timestamps, author names and comments. Document logs with similar structure can be derived from different SCM systems, such as *CVS*, *Subversion*, *SourceSafe*, *Clear Case* and others.

When we deal with the document logs, we have to answer the following questions: how to identify the cases (process instances), how to identify the document types, how to abstract from the details of the log, and how to ignore unnecessary information. For many software projects, a *case* corresponds to a subproject, a plugin or a special repeatable phase of product development. In our example, a case corresponds to a project. We detect the *documents types* by

detecting the similarity of paths and names, see Sect. 4.1 for details[3]. The same technique is used for abstracting from the log details and for ignoring the noise, i.e. ignoring the exceptional infrequent commits. However, the latter problems should be treated both on the log level and on the algorithm level, see Sect. 4.2.

Now, after introducing the structure of document logs, we state the problem: *Multi-perspective Software Process Mining from the Document Logs of SCM systems. Multi-perspective* means that we deal not only with control-flow, but also with organizational and informational perspectives, since the information about them is also available in the document logs (documents and authors).

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 organizational and the informational models from the *event logs*. The events in these logs correspond to process *activities* produced by Workflow Management Systems (WfMS), Enterprise Resource Planning (ERP) systems or other systems providing process support. 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 Product Data Management (PDM). All the presented algorithms are integrated as plugins to the *ProM* tool [21], which is described at the end of this section.

4.1 Abstraction on the Log Level

As it was already pointed out in Sect. 3.2, 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 it *abstraction on the log level*. The ProM tool contains a set of *filters*, which helps us solving this problem. One of the most mature filters in this context is the remap filter.

The remap filter supports mapping the names of documents from the document log to the abstract names. With the help of regular expressions, we specify the paths that should be mapped to the abstract names. For example, if the path contains "/models/", the filename contains "design" and has extension ".mdl", then it could be mapped to "DES". In the similar way all the ".java" files are mapped to "CODE", test plans – to "TEST", review documents – to "REV" and verification results – to "VER". Table 2 shows the result of this filter applied to the log of Table 1. Additionally, if we want to ignore test plans, for example, then we map corresponding regular expression to an empty name.

[3] Theoretically, detecting the document types is a challenging task, which requires additional information, which is not available in the logs; an idea was proposed in our paper [30].

Table 2. Filtered Software Log

Document	Date	Author
DES	01.01.05 14:30	designer
CODE	01.01.05 15:00	developer
TEST	05.01.05 10:00	qaengineer
REV	07.01.05 11:00	manager
DES	01.02.05 11:00	designer
TEST	15.02.05 17:00	qaengineer
CODE	20.02.05 09:00	developer
REV	28.02.05 18:45	designer
DES	01.03.05 11:00	designer
VER	15.03.05 17:00	qaengineer
CODE	20.03.05 09:00	designer
REV	28.03.05 18:45	manager

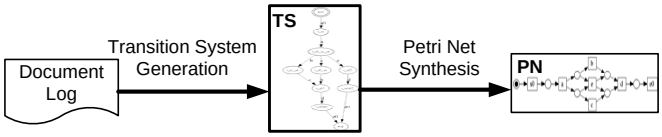

Fig. 2. Generation and Synthesis Approach Scheme

Thus, using this filter, we solve the problem of documents of the same type that have different names in different projects and provide the possiblity to make an abstraction from the given log.

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). In our example, the log contains three cases: $\langle DES, CODE, TEST, REV \rangle$, $\langle DES, TEST, CODE, REV \rangle$, and $\langle DES, VER, CODE, REV \rangle$.

Generation and Synthesis Approach The approach presented in this section is a *two-step approach* (see Fig. 2): (Step 1) it takes a document log and generates a transition system (TS) from it; (Step 2) it synthesizes a Petri Net (PN) from the transition system.

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 it "clever" transition system generation or *abstraction on the model level*. Since software logs usually do not contain all possible traces and, thus, represent only a part of a possible behaviour, and since they contain a lot of unnecessary details that must be ignored, we should use the generation strategies to resolve these issues. 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* [17] are used for transforming transition systems (state-based specification) to Petri nets (event-based specification), which are more compact. Furthermore, the whole set of well-developed process analysis and verification techniques from the area of Petri nets are available for the generated models; then, the Petri net models can be converted to EPCs and other formalisms supported by different existing Process Management Systems.

The transition system shown in Fig. 3 is *constructed* from the log given in Table 2. In this example, a *state* is defined as a *set* of documents representing the complete history of a case at a point of time. For example, for the first case, there are the following states: $\{\}$, $\{DES\}$, $\{DES, CODE\}$, $\{DES, TEST, CODE\}$ and $\{DES, TEST, CODE, REV\}$. 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. 4. This Petri net has the same behaviour as the TS, events of the TS correspond to the transitions of the PN.

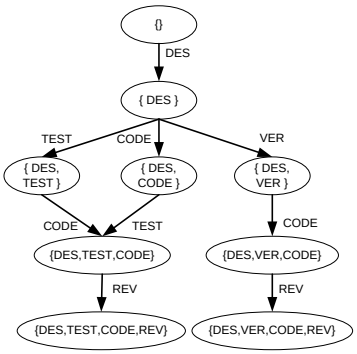

Fig. 3. Generated Transition System

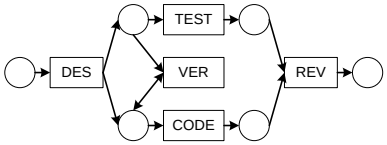

Fig. 4. Synthesized Petri Net

In general, we can construct transition systems in different ways by means of defining a state not only as a set, but as a *multi-set* or a *sequence* of documents and look not only at the complete history, but also at the complete future or *partial* future and history of a case. Next, we can *modify* the constucted TS using some strategy. The result of the modification with the "Extend Strategy" is shown in Fig. 5. Basically, this strategy makes 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. The motivation for the "Extend" strategy is that it in many cases it

is unrealistic that all possible interleavings of activities are actually present in the log. As a result, we generated an additional transition VER from state $\{DES, CODE\}$ to state $\{DES, VER, CODE\}$. A Petri net corresponding to this TS is shown in Fig. 6. This Petri net is more general then the first one, it allows an additional trace, namely $\langle DES, CODE, VER, REV \rangle$.

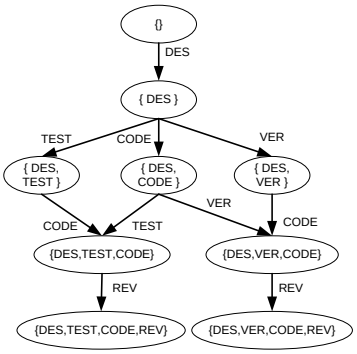

Fig. 5. Extended Transition System

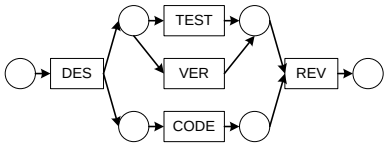

Fig. 6. Synthesized PN for Extended TS

The first ideas of the generation and synthesis approach were presented in our previous paper [31], then the algorithms were significantly improved and successfully implemented in the context of ProM; the tool Petrify [16] is used in the synthesis phase. The ProM tool includes implementations of different strategies, which allow us to abstract from the information given in the log and to build understandable and analyzable process models. This approach overcomes many limitations of the traditional process mining approaches. It can deal with *complicated process constructs* (for example, non-free-choice constructs and loops) and *duplicates* (documents of different types with the same names); it can overcome *overfitting* (generated model allows only for the exact behaviour seen in the log) and *underfitting* (model overgeneralizes the things seen in the log) and it produces *consistent models*. However, by now, this approach can hardly deal with *noise* (incorrectly logged events and exceptions); so, the other approaches that treat this problem, are presented in the next Section.

Other Approaches for Control Flow Mining The generation and synthesis approach introduced in the previous section is particularly suitable for the situation at hand, as it can relate the availability of documents in the system to the states of the development process. In the process mining domain a number of further algorithms for control flow mining have already 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* [8] can also derive a Petri net model from an event log, however it is based on analyzing the immediate successor relation between event types, i.e. documents. Another algorithm, the *Multi-phase approach* [20], 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. The Multi-phase approach is very robust, as it can map fuzzy branch and join situations in a process model (i.e., where it is hard to find out whether the branch or join has AND- or XOR-semantics) to the OR-connector of the EPC language.

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 this property is useful to determine what *exactly* has happened in the development process, there are also situations where one wants to abstract from infrequent behavior, i.e. *noise*. The ability to abstract from noise is particularly required in the context of large and semi-structured processes, which are characterized by a large number of involved tasks with a high ratio of ordering relations between them. Mining logs from such processes with precise algorithms usually results in "spaghetti-like" process models, which are hard to derive high-level information from.

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* [39] can do so by employing a sophisticated heuristics which, based on the frequency of discovered ordering relations, attempts to discard exceptional behavior. Another approach in this direction is the *Genetic Miner* [4]. 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. Specifically designed mutation operators and fitness metrics enable the Genetic Miner to gradually approach the optimal process model.

4.3 Mining other perspectives

The control flow, i.e. the ordering of tasks within a process, is only one *perspective* addressed in this paper. Additional information, such as the timestamp of an event or its originator (i.e., the person having triggered its occurrence) can be used to derive high-level information about the software process also in other perspectives.

Resource Perspective One perspective different from control flow is the *resource perspective*, which looks at the set of people involved in the process, and their relationships. The *Social Network Miner* [5] for example can generate the *social network* of the organization, which may highlight different relationships between the persons involved in the process. One example of a social network represents the *handover of work* between the resources involved in the process. The resources are symbolized by nodes, while each arc represents that at least once a work was passed in that direction, i.e. these persons subsequently worked

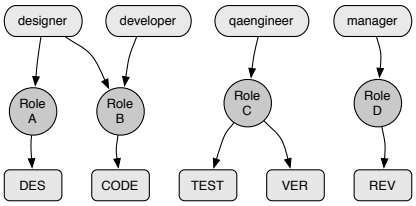

Fig. 7. Example of an organizational structure

on the same project. There are also social networks highlighting other relationships, e.g. *subcontracting*, where an event from one person is encompassed by two events from another person.

The *Organizational Miner* also addresses the resource perspective, attempting to cluster resources which perform similar tasks into *roles*. An example of mining this organizational structure from the log in Table 2 is shown in Figure 7. Based on the overlap of subsets of resources having executed each task, four roles have been derived. A role can both be required for a number of different tasks (e.g., "Role C" can both handle tasks "TEST" and "VER"), and resources may occupy several roles (e.g., the resource "designer" has both "Role A" and "Role B". 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, for example, the test phase is highlighted as the point in the process where most time is spent, it may be helpful to assign more staff to this task.

Information Perspective As it has already been mentioned in Section 4.1, it is helpful to abstract from low-level events in the log. However, there may also be situations where the exact composition of higher-level modules corresponding to development phases is not known precisely. The *Activity Miner* [25] addresses this problem, which is common due to the low-level nature of most logs. It 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, can be helpful for unveiling hidden dependencies between documents, or for a re-structurization of the document repository layout.

4.4 Process Analysis and Verification

The mining approaches described in the previous sections mainly serve the purpose to extract high-level information from a process enactment log. This 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 not so interesting how exactly the process is executed, but rather if this 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* [35], which takes an enactment log and an associated process model, e.g. a Petri net, as input. The goal is to analyze the extent to which the process execution, as recorded in the log, corresponds to the given process model. Also, conformance checking can point out the parts of the process where the log does not comply, and the process instances which are deviant. This technique can be used both for process verification (i.e., is the actual execution compliant to my defined development process?) and for process analysis (i.e., where does my organization fail to comply with the defined process?). In the context of strictly defined development processes, e.g. in CMMI or government-sponsored development, the hard proof of compliance to these processes can be a competitive advantage.

Another technique to this end is *LTL Checking* [3], which analyzes the log for compliance with specific constraints, where the latter are specified by means of linear-temporal logic (LTL) formulas. An example of such a constraing is: "A testing activity must be performed only after the design activity is finished". LTL checking can be used to verify these constraints in a log, and to pinpoint the specific cases which do not comply. Regarding the example log in Table 2, all three cases satisfy the above constraint. 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, isolated corporate guidelines or best development practices.

While the ProM framework supports all of the above process mining techniques, it also features techniques for process model analysis and verification in the absence of a log. Advanced process model analyzers, such as *Woflan* [38], which is also integrated in ProM, can check e.g. a Petri net model for *deadlocks* (i.e., potential situations in which execution will be stuck), or verify that there exists a valid *place invariant* (i.e., all process executions will complete properly with no enabled task left behind). For the software process models shown in Figure 4 and Figure 6, Woflan detected no deadlocks or other anomalies. 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 [21] 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. Information as shown earlier in tabular form can be stored in MXML. One MXML file can store information about multiple processes. Per process events related to particular process instances (often called 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 originator (who executed the activity; in this paper often referred to as the "author"), timestamps (when did the event occur), 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 is been used to develop many organization/system-specific conversions (e.g., hospitals, banks, governments, etc.). As will be shown later, the ProMImport Framework can also be used to extract event logs from systems such as Subversion and CVS (Concurrent Versions System).

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 so-called "plug-ins" that implement a specific mining approach. Although the most interesting plugins in the context of this paper are the mining plugins, it is important to note that there are in total five types of plug-ins:

Mining plug-ins which implement some mining algorithm, e.g., mining algorithms that construct a Petri net based on some event log.

Export plug-ins which 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 which implement an "open" functionality for exported objects, e.g., load instance-EPCs from ARIS PPM.

Analysis plug-ins which 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 which implement conversions between different data formats, e.g., from EPCs to Petri nets and from Petri nets to YAWL and BPEL.

The next section will illustrate the application of some of these plug-ins. However, since there are currently more than 140 plug-ins it is impossible to give a representative overview. One of these more than 140 plug-ins is the mining plug-in that generates the transition system that can be used to build a Petri net model. Note that for this particular approach ProM calls Petrify [17] to synthesize the Petri net.

5 Evaluation and Applications

As an evaluation example, we decided to take the *ArgoUML* project. ArgoUML is a popular open-source UML modeling tool. It is an open source development project (BSD license), which provides access to its source files maintained with the *Subversion* SCM system. ArgoUML is organized as a set of *subprojects* with separate members lists and goals, but with the *same file organization* and the same development tools. Thus, different subprojects use the same naming conventions and development rules, it significantly simplifies the work of process mining algorithms.

We decided to take a look at five subprojects, where the ArgoUML support for the following languages is developed: $C++$, $C\#$, IDL, PHP and *Ruby*. Thus, all these projects correspond to *cases* (process instances) and the overall mined model can be called "the process model for developing language support for ArgoUML". So, our goal is: (1) to derive a formal *plausible software development process model* (control-flow perspective) from the document logs; (2) to enhance the resulting model with the performance and the organization perspectives; (3) to apply the process analysis and verification techniques.

First, using the *svn log* utility provided by Subversion, we generated logs for all the five subprojects. Then, using the *ProMImport* tool, the logs were converted to the MXML format, which is accepted by the *ProM* tool; all the logs were merged to a single log containing one process with five process instances containing about 400 commits. A small fragment of the log is shown in Fig. 8.

```
<AuditTrailEntry>
    <WorkflowModelElement>/trunk/www/index.html</WorkflowModelElement>
    <EventType>complete</EventType>
    <Timestamp>2006-06-02T19:49:16.000+01:00</Timestamp>
    <Originator>tfmorris</Originator>
</AuditTrailEntry>
<AuditTrailEntry>
    <WorkflowModelElement>/trunk/src/org/argouml/language/cpp
                          /ui/SettingsTabCpp.java</WorkflowModelElement>
    <EventType>complete</EventType>
    <Timestamp>2006-06-02T20:28:40.000+01:00</Timestamp>
    <Originator>mvw</Originator>
</AuditTrailEntry>
```

Fig. 8. A log fragment.

The resulting log contains project specific paths and different commits, which are not relevant for the software process. Therefore, using the *remap filter*, we replaced project specific paths with the abstract names. In our example, all the commited documents (files) containing "/src/" in their paths with ".java" at the end were mapped to "SRC", all the "readme.*" files – to "README", all the files

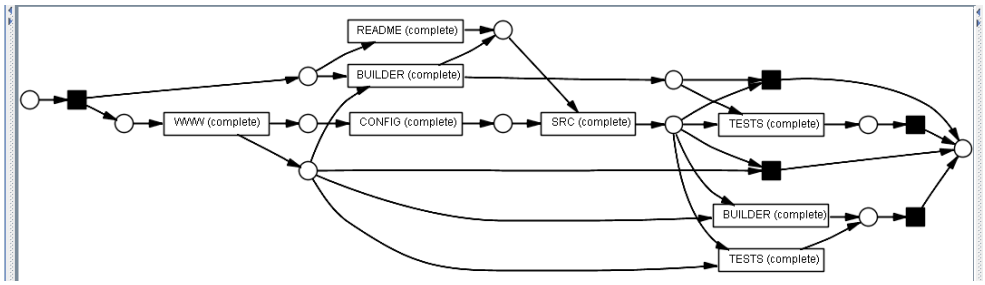

Fig. 9. Petri Net for the ArgoUML Project

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. It should be noted that this mapping corresponds to the naming conventions of the ArgoUML project. Thus, at the end we received an abstract log, which can be processed by our algorithms.

After executing the algorithms of the *generation and synthesis* approach, we obtained the Petri net shown in Fig. 9. Here, for the sake of readability, we show this simplified Petri net without loops; we use to derive acyclic processes by means of applying the "Kill Loops" modification strategy to the transition system and synthesizing a Petri net from it. Thus, the Petri net focuses on the starting 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 is changed after writing code. Now, since we have a model, it can be extended for dealing with time and for representing the statistical data about the duration of tasks.

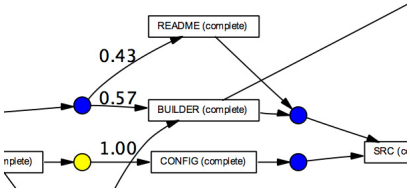

Fig. 10. Performance Analysis for the ArgoUML Project

The Petri net model of the development process can now be used for enhanced analysis within the ProM framework. Figure 10 shows the result of a *performance analysis* based on the mined model and the log. The states, i.e. places, have 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 their respective probability (i.e., the fraction of cases for which this path was taken).

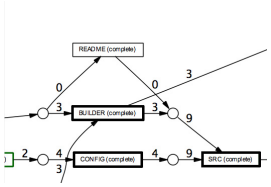

Fig. 11. Conformance Analysis for the ArgoUML Project

Further, a *conformance analysis* can be performed using the Petri net model and the associated log. Figure 11 shows the *path coverage* analysis of the conformance checker. All activities that have been executed in a specific case (or, set of cases) are decorated with a bold border, and arcs are annotated with the frequency they have been followed in the case. In this example, it can be seen that "README" was not executed for the "CPP" case, i.e. the C++ language support team has not created a README file.

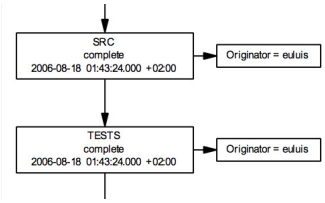

Fig. 12. LTL Analysis for the ArgoUML Project

One constraint in a software development project could be, that developers working on the source code should not write tests as well. Figure 12 shows the result of checking a corresponding *LTL* formula on the ArgoUML log. In the C++ language support case, which is shown in Figure 12, both source code and tests have been submitted by the developer "euluis", thereby violating this constraint.

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 13 illustrates the hand-over

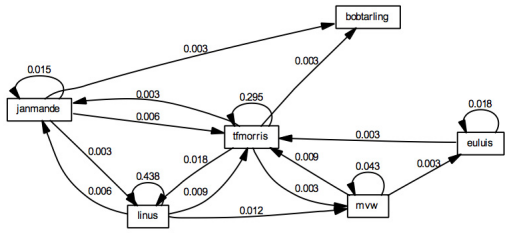

Fig. 13. Social Network for the ArgoUML Project

of work between ArgoUML developers. It shows that some developers are only involved in specific phases of the project (e.g., "bobtarling" appears to only work 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 at hand one may prefer different collaboration patterns, which can be checked conveniently in a mined social network like this.

Thus, in this section, we presented a small handy example of the real software project, where a subset of the big set of process mining and analysis techniques supported by ProM was applied.

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 ArgoUML since this is a larger project where the repository is freely available. 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 EIT, NWO, the University of Paderborn and International Graduate School of Dynamic Intelligent Systems, and the Technology Foundation STW, applied science division of NWO and the technology programme of the Dutch Ministry of Economic Affairs.

References

1. *MSR 2004: International Workshop on Mining Software Repositories*, Washington, DC, USA, 2004. IEEE Computer Society.

2. *MSR 2005 International Workshop on Mining Software Repositories*, New York, NY, USA, 2005. ACM Press.
3. W.M.P. van der Aalst, H.T. de Beer, and B.F. van Dongen. Process Mining and Verification of Properties: An Approach based on Temporal Logic. BETA Working Paper Series, WP 136, Eindhoven University of Technology, Eindhoven, 2005.
4. W.M.P. van der Aalst, A.K. Alves de Medeiros, and A.J.M.M. Weijters. Genetic Process Mining. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536, pages 48–69, 2005.
5. W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering Social Networks from Event Logs. *Computer Supported Cooperative work*, 14(6):549–593, 2005.
6. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
7. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
8. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
9. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
10. Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.
11. J.E. Cook, Z. Du, C. Liu, and A.L. Wolf. Discovering models of behavior for concurrent workflows. *Computers in Industry*, 53(3):297–319, 2004.
12. J.E. Cook, C. He, and C. Ma. Measuring Behavioral Correspondence to a Timed Concurrent Model. In *Proceedings of the 2001 International Conference on Software Mainenance*, pages 332–341, 2001.
13. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
14. J.E. Cook and A.L. Wolf. Event-Based Detection of Concurrency. In *Proceedings of the Sixth International Symposium on the Foundations of Software Engineering (FSE-6)*, pages 35–45, 1998.
15. J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.
16. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, 1997.
17. Jordi Cortadella, Michael Kishinevsky, Luciano Lavagno, and Alexandre Yakovlev. Deriving Petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882, 1998.
18. Davor Cubranic and Gail C. Murphy. Hipikat: recommending pertinent software development artifacts. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 408–418, Washington, DC, USA, 2003. IEEE Computer Society.

19. Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communication of the ACM*, 35(9):75–90, 1992.
20. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.W. Ling, editors, *International Conference on Conceptual Modeling (ER 2004)*, volume 3288, pages 362–376, 2004.
21. B.F. van Dongen, A.K. Alves de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Application and Theory of Petri Nets 2005*, volume 3536, pages 444–454, 2005.
22. W. Droschel and H. Wiemers. *Das V-Modell 97, Der Standard für die Entwicklung von IT-Systemen mit Anleitung für den Praxiseinsatz*. Oldenbourg, 2000.
23. Peter H. Feiler and Watts S. Humphrey. Software process development and enactment: Concepts and definitions. Technical Report CMU/SEI-92-TR-004, SEI Carnegie Mellon, 1993.
24. Volker Gruhn. Process-centered software engineering environments - a brief history and future challenges, 2002.
25. C.W. Günther and W.M.P. van der Aalst. Mining Activity Clusters from Low-level Event Logs. BETA Working Paper Series, WP 165, Eindhoven University of Technology, Eindhoven, 2006.
26. Federico Iannacci. Coordination Processes in Open Source Software Development: The Linux Case Study. http://opensource.mit.edu/papers/iannacci3.pdf, apr 2005.
27. Ivar Jacobson, Grady Booch, and James Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
28. Ekkart Kindler, Vladimir Rubin, and Wilhelm Schäfer. Incremental Workflow mining based on Document Versioning Information. In Mingshu Li, Barry Boehm, and Leon J. Osterweil, editors, *Proc. of the Software Process Workshop 2005, Beijing, China*, volume 3840 of *LNCS*, pages 287–301. Springer, May 2005.
29. Ekkart Kindler, Vladimir Rubin, and Wilhelm Schäfer. Activity mining for discovering software process models. In B. Biel, M. Book, and V. Gruhn, editors, *Proc. of the Software Engineering 2006 Conference, Leipzig, Germany*, volume P-79 of *LNI*, pages 175–180. Gesellschaft für Informatik, March 2006.
30. Ekkart Kindler, Vladimir Rubin, and Wilhelm Schäfer. Incremental Worklfow Mining for Process Flexibility. In *Proc. of the Seventh CAiSE'06 Workshop on Business Process Modeling, Development, and Support (BPMDS'06), Luxembourg*, jun 2006.
31. Ekkart Kindler, Vladimir Rubin, and Wilhelm Schäfer. Process Mining and Petri Net Synthesis. In Johann Eder and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *LNCS*. Springer, 2006.
32. Keir Mierle, Kevin Laven, Sam Roweis, and Greg Wilson. Mining student cvs repositories for performance indicators. In *MSR '05: Proceedings of the 2005 international workshop on Mining software repositories*, pages 1–5, New York, NY, USA, 2005. ACM Press.
33. A. Mockus, R.T. Fielding, and J. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Trans. Software Engineering and Methodology*, 11(3):309 – 246, 2002.
34. L Osterweil. Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering*, pages 2–13, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

35. A. Rozinat and W.M.P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In C. Bussler et al., editor, *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812, pages 163–176, 2006.
36. Robert J. Sandusky, Les Gasser, and Gabriel Ripoche. Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community. In *MSR 2004: International Workshop on Mining Software Repositories*, 2004.
37. Walt Scacchi. Understanding the requirements for developing open source software systems. *IEE Proceedings - Software*, 149(1):24–39, 2002.
38. H.M.W. Verbeek and W.M.P. van der Aalst. Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825, pages 475–484, 2000.
39. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.
40. Thomas Zimmermann and Peter Weissgerber. Preprocessing cvs data for fine-grained analysis. In *Proc. 1st International Workshop on Mining Software Repositories (MSR)*, may 2004.