Process Mining of Test Processes: A Case Study

A. Rozinat[1], I.S.M. de Jong[2], C.W. Günther[1], and W.M.P. van der Aalst[1]

[1] Department of Information Systems, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
{a.rozinat,c.w.gunther,w.m.p.v.d.aalst}@tue.nl
[2] ASML, P.O. Box 324, NL-5500 AH, Veldhoven, The Netherlands
ivo.de.jong@asml.com

Abstract. Process mining techniques attempt to extract non-trivial and useful information from event logs. For example, there are many process mining techniques to automatically discover a process model describing the causal dependencies between activities. Moreover, using conformance checking it is possible to investigate and quantify deviations between the real process and the modeled process. Several successful case studies have been reported in literature, all demonstrating the applicability of process mining. However, these case studies refer to rather structured administrative processes. In this paper, we investigate the applicability of process mining to less structured processes. We report on a case study where the ProM framework has been applied to the test processes of ASML (the leading manufacturer of wafer scanners in the world). This case study provides many interesting insights. On the one hand, process mining is also applicable to the less structured processes of ASML. On the other hand, the case study also shows the need for alternative mining approaches able to better visualize processes and provide more insights.

1 Introduction

ASML is the world's leading manufacturer of chip-making equipment and a key supplier to the chip industry. ASML designs, develops, integrates and services advanced systems to produce semiconductors. In short, it makes the wafer scanners that print the chips. These wafer scanners are used to manufacture semi-conductors (e.g., processors in devices ranging from mobile phones ad MP3 players to desktop computers). Wafer scanners are complex machines consisting of many building blocks and use a photographic process to image nanometric circuit patterns onto a silicon wafer, much like a camera prints an image on film. Because of competition and fast innovation, the time-to-market is very important. There is an ongoing effort to reduce the line widths on silicon wafer to enhance the performance of the manufactured semi-conductors. Every new generation of wafer scanners is balancing on the border of what is technologically possible. As a result, the testing of manufactured wafer scanners is an important but also time-consuming process. Every wafer scanner is tested in the factory of ASML. When it passes all tests, the wafer scanner is disassembled and shipped to the customer where the system is re-assembled. At the customer's site, the

wafer scanner is tested again. Clearly, testing is a time-consuming process and takes several weeks at both sites. Since time-to-market is very important, ASML is involved in an ongoing effort to reduce the test period. To assist ASML in these efforts, we applied process mining techniques to their test processes.

The basic idea of *process mining* is to discover, monitor and improve *real* processes (i.e., not assumed processes) by extracting knowledge from event logs. Today many of the activities occurring in processes are either supported or monitored by information systems. Consider for example ERP, WFM, CRM, SCM, and PDM systems to support a wide variety of business processes while recording well-structured and detailed event logs. However, process mining is not limited to information systems and can also be used to monitor other operational processes or systems. For example, we have applied process mining to complex X-ray machines, high-end copiers, web services, careflows in hospitals, etc. All of these applications have in common that *there is a notion of a process* and that *the occurrences of activities are recorded in so-called event logs*. Assuming that we are able to log events, a wide range of *process mining techniques* comes into reach. The basic idea of process mining is to learn from observed executions of a process and can be used to (1) *discover* new models (e.g., constructing a Petri net that is able to reproduce the observed behavior), (2) check the *conformance* of a model by checking whether the modeled behavior matches the observed behavior, and (3) *extend* an existing model by projecting information extracted from the logs onto some initial model (e.g., show bottlenecks in a process model by analyzing the event log). All three types of analysis have in common that they assume the existence of some *event log*.

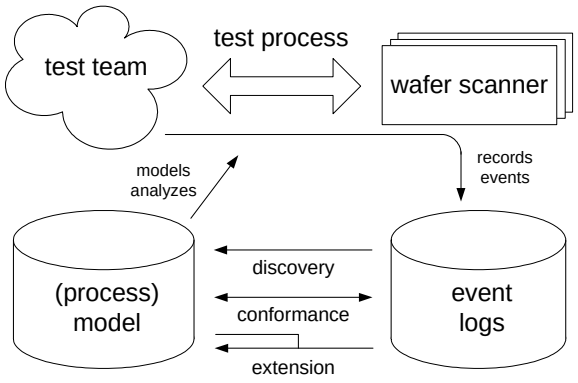


Fig. 1. Based on the event logs of the wafer scanners, three classes of process mining techniques are used: (1) "discovery", (2) "conformance", and (3) "extension"

At any point in time, ASML's wafer scanners record events that can easily be distributed over the internet. Hence, any event that takes place during the test process can be recorded easily. The availability of these event logs and the

desire of ASML to improve the testing process, triggered the case study reported in this paper. In collaboration between Eindhoven University of Technology and ASML, we analyzed the testing process using the three classes of process mining techniques mentioned before: (1) "discovery", (2) "conformance", and (3) "extension" (cf. Figure 1). Using process discovery, we tried to answer the question "How are the tests actually executed?", i.e., based on the event logs we automatically constructed process models showing the ordering and frequency of test activities. This revealed that the real process is much more complicated than the idealized reference model that ASML is using to instruct the test teams. The reference model shows a rather structured process while in reality the testing process requires much more flexibility. Using conformance checking techniques, we investigated this further by answering the question "How compliant are the actual test executions to the reference process?". Through conformance checking we were able to quantify and pinpoint the deviations of the real test process from the idealized reference model. Finally, we used process mining to answer the question "Where is the most time spent in the test process?". For this we extended the discovered models and the reference model with additional information extracted from the logs. This way, it was possible to project performance bottlenecks directly on the process model.

In this paper, we show that recently developed process mining techniques can be used to answer the three questions stated above. This is interesting since, so far, process mining has only been applied to rather structured processes (e.g., administrative processes supported through some information system [3]). For the case study we used our *ProM framework*. ProM is open source and uses a plug-able architecture, e.g., developers can add new process mining techniques by adding plug-ins without spending any efforts on the loading and filtering of event logs and the visualization of the resulting models. Version 4.0 of ProM provides 142 plug-ins. For example, there are more than 15 plug-ins to discover process models from event logs. As we will see, it is not easy to apply traditional process mining techniques to less structured processes such as the testing of wafer scanners. Therefore, this paper also discusses the limitations of existing techniques and suggests some alternative approaches. In fact, two new plug-ins have been developed based on this case study.

The remainder of this paper is organized as follows. Section 2 reviews related work both in process mining and the test process optimization domains. Next, the context of the case study is described in more detail in Section 3. Section 4 presents the results of this study, and concrete improvement actions for the ASML test process are proposed in Section 5. Future challenges for process mining and the two new ProM plug-ins are discussed in Section 6. Section 7 concludes the paper.

2 Related Work

In this section, we first review related work on process mining and then review related work on test processes.

Since the mid-nineties several groups have been working on techniques for process mining [5, 6, 12, 15, 19, 17, 31], i.e., discovering process models based on observed events. In [4] an overview is given of the early work in this domain. The idea to apply process mining in the context of workflow processes was introduced in [6]. In parallel Datta [15] looked at the discovery of business process models. Cook et al. investigated similar issues in the context of software engineering processes [12]. Herbst [24] was one of the first to tackle more complicated processes, e.g., processes containing duplicate tasks.

Most of the classical approaches have problems dealing with concurrency. The α-algorithm [5] is an example of a simple technique that takes concurrency as a starting point. However, this simple algorithm has problems dealing with complicated routing constructs and noise (like most of the other approaches described in literature). In [19, 17] a more robust but less precise approach is presented. Heuristics [31] or genetic algorithms [2, 16] have been proposed to deal with issues such as noise. In the case study we used ProM's `Heuristic Miner` [31] to discover suitable process models.

For conformance checking we used the `Conformance Checker` [28] and the `LTL Checker` [1]. Conformance checking as used in this paper is closely related to the work of Cook et al. [13, 11] who have introduced the concept of process validation. They propose a technique comparing the event stream coming from the process model with the event stream from the execution log based on two different string distance metrics.

Process mining can be seen in the broader context of Business Process Intelligence (BPI), Business Activity Monitoring (BAM), and business process visualization [30]. In [20, 29] a BPI toolset on top of HP's Process Manager is described. The BPI toolset includes a so-called "BPI Process Mining Engine". In [27] Zur Muehlen describes the PISA tool which can be used to extract performance metrics from workflow logs. Similar diagnostics are provided by the ARIS Process Performance Manager (PPM) [25]. It should be noted that BPI tools typically do not allow for process discovery and conformance checking, and offer relatively simple performance analysis tools that depend on a correct a-priori process model.

After providing an overview of process mining literature, we discuss related work on the improvement of test processes. Most test process optimization techniques are currently focused on optimizing the test organization instead of the processes itself. If the test process is optimized by technology-based optimization techniques, then these techniques are either only applicable in a single discipline or still very general. An example of detailed mono-disciplinary optimization techniques is the use of test coverage measures [26] for software testing. These detailed measures are used to determine the coverage of a set of test cases on the system under test. These methods are applied for small software systems where high quality levels are important. These methods do not scale very well for large software systems.

More general test process optimization techniques apply risk-based test sequencing [7, 22] in combination with a certain stop criterion such that the test

cases which cover the highest risk (high level test cases) are executed first. The disadvantage of this approach is that these high level test cases are often inconclusive about the root cause of a failure. A lengthy diagnosis action is required when these test cases fail. A sequencing algorithm which optimizes a test phase including diagnosis and fix actions is described by [10, 9]. The system test model which is used in [10, 9] models a test problem independent of the discipline.

The case study reported in this paper investigates the differences between the actual, executed test sequences and the planned test sequences. The next section, introduces the case study. Then, we start answering the three questions stated in the introduction. First, we discover the actual process and then we compare the reference model with the real test sequences. The differences between the actual and the planned test sequences are then used to suggest improvements of the test process (or ASML's organization).

3 Case Study

This section introduces the case study where process mining was applied to the test process of ASML's wafer scanners. After providing some background information (Section 3.1), we describe the test process of a wafer scanner in more detail (Section 3.2), and then look at the log data recorded during these tests (Section 3.3). The event logs serve as input for our process mining techniques and the results of their analysis are described in Section 4.

3.1 ASML's Wafer Scanners

Nowadays, semi-conductors can be found in many appliances around us. Mobile phones, MP3-players, television sets and desktop computers contain processors and computer memory. These semi-conductors are manufactured in twenty-plus steps, called the semi-conductor manufacturing process. Images of a transistor pattern are placed on a silicon wafer with typical line widths of 90 nm and less. This imaging process is repeated up to 30 or more times to form a completely functional integrated circuit. The imaging of the pattern on a silicon wafer is done by a so-called wafer scanner. The semi-conductor manufacturing process is explained in detail in [14]. The case study was conducted with ASML. ASML is the leading manufacturer of wafer scanners in the world.

A wafer scanner consists of around one thousand building blocks. These building blocks can be considered a system in itself; they can consist of an entire electronics rack, thousands of lines of code, or a complete lens system. Most of these building blocks are manufactured at suppliers. Together, these building blocks form a scanner. After assembly of the building blocks, the wafer scanners are calibrated and tested. The calibration and test sequence of a wafer scanner ends with a system qualification phase. In this system qualification phase, the system performance in terms of throughput, overlay and imaging performance is measured. The throughput of a wafer scanner is a measure of the wafer production speed. The overlay of a wafer scanner is a measure of how accurate the different

patterns are placed on top of each other in each next layer. The imaging performance of a wafer scanner determines the line width of the structures in the integrated circuit. The capabilities of a TWINSCAN^{TM} XT:1900Gi wafer scanner in terms of throughput, overlay and imaging performance are resp. $\geq$ 131 wph, $\leq$ 6 nm and $\leq$ 40 nm [8]. Moore's law[3] is forcing companies like ASML to constantly innovate and build machines that can handle silicon wafers with smaller line widths. This requires a continuous balance between performance (smaller line widths) and reliability (semi-conductors that can be produced without failures). Therefore, ASML is working on the boundaries of what is technologically possible and, hence, it is important to have good test processes in place.

3.2 The Test Process

The whole test process consists of three phases: (1) the calibration phase, (2) the test phase (the actual testing), and (3) the final qualification phase. The whole process typically takes a few weeks. When finished, the wafer scanner is partly taken apart and shipped to a customer. A part of the calibration and test phase is repeated at the customer site, after re-assembling the wafer scanner.

Why is this test process so important for ASML? ASML operates in a market where the time-to-market of system enhancements and the time-to-market of new system types is critical. Wafer scanners are continuously enhanced. As a result, the number of manufactured wafer scanners of a single type is typically less than 50. And with each new type, parts of the calibration and test phase are adjusted. On average five different system types are manufactured in parallel. The short time-to-market, the constant innovation, and the high value of wafer scanners make testing very important. Spending too much time on testing will result in high inventory costs and lost sales. However, inadequate tests will result in systems which are malfunctioning.

Sets of calibration and test actions are grouped into so called *job steps*. These job steps are executed according to a certain sequence. Only large changes in the system design result in changes in the job step sequence, so the job step sequence can be considered a fixed sequence across different systems. Some of these jobsteps can be executed independently of each other. An example sequence of three job steps is depicted in Figure 2. Note that in ASML such structures are referred to as "sequences". However, strictly speaking these are process models rather than sequences. The synchronization point *sync* forces that both job step A and job step B must be finished before job step C can start. Each calibration action or test case can fail. Some of the causes for test failure can require a replacement of a faulty hardware component. The duration of this replacement can take up to hours or longer. If such a failing test is in the example job step A, then the independent job step B can be started to ensure maximal progress.

[3] Moore (founder of Intel), commenting on the growth of the microelectronics industry in 1964, noted a doubling of the number of elements on a produced chip once every 12 months. For a decade that meant a growth factor of approximately 1000. Today, when Moore's Law is quoted, the time constant typically quoted is 18 months.

When the replacement hardware becomes available, either job step B is finished first and then job step A is finished, or the other way around. If job step B was already finished, then the hardware is replaced and job step A is finished. Job step C is started when job step A and B are both finished. Note that a failure in a test case in job step C results in no activity on the system (idle time) until the malfunctioning part of system is fixed and testing can continue.

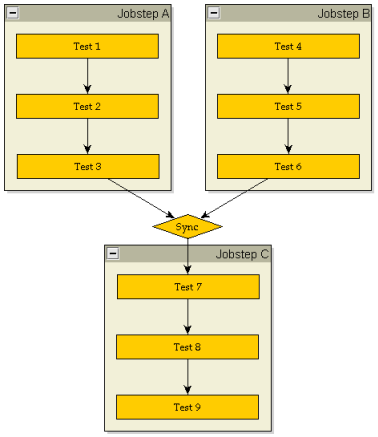


Fig. 2. Example sequence of three job steps with a synchronization point

Some of the causes for a failure can be fixed immediately. For example, some parameters in the system can be out of specification. This measurement information can now be used to adopt the control set-points in the system. After a second measurement, the parameter can be within specification and the test passes. Most of the software which executes the tests is constructed such that this fast-fix loop is automated. Testing, calibration and retesting is performed in a single test.

Finally, a change in low-level machine parameters, because of a hardware replacement, can cause a re-execution of a previous job-step. For instance, the profile of some of the mirrors in a wafer scanner are measured and stored in X,Y and Z directions. This profile information is used in all positioning calibrations, such that the error caused by the non-flat mirrors are minimized. Replacing these mirrors results in a new profile. For this reason, a large set of job steps needs to be redone if a faulty mirror is replaced in one of the last job steps in the sequence.

In summary, job steps are executed according to a fixed sequence for a set of machine types. The sequence allows variation of the detailed tests within the limits of the synchronization points. Next to that, the actual execution of tests results in failing test cases. These failing test cases can result in a lengthy re-test of parts of the sequence depending on the failure at hand.

For ASML, the goal is to minimize the waiting time for a hardware fix (idle time) and to reduce the re-execution of parts of the job-step sequence. This goal could be easily met by testing all components and building blocks thoroughly before and during system assembly. However, the increase in test effort would result in an increase of the total test duration and therefore an increase in time-to-market. This is the main reason that testing everything thoroughly beforehand is not considered a solution, i.e., the main goal is a reduction of the duration of the test process and not cutting costs.

The work presented in this paper attempts to shorten the test process by applying process mining techniques to the existing test processes, i.e., we analyze the test process based on historical data to find bottlenecks and ideas for improvement. Historical data helps to provide insight into where the *real* bottlenecks are, and, therefore, enables *specific* actions to improve future test executions. For example, failing tests which are often followed by idle time can be detected and addressed. Furthermore, the executed test sequences can be compared to the given *reference sequence*, i.e., the "ideal process model" defined by ASML. Discrepancies can be related to the re-executed parts of the sequence. For example, failing tests which often cause a re-execution of a part of the sequence can be addressed and improved.

The next subsection describes the logging characteristics, i.e., the form historical data is recorded during the test process. Moreover, we show how this log data can be converted to apply process mining techniques in the ProM framework [18].

3.3 Log Data and Conversion

Each wafer scanner in the ASML factory produces a log of the software tests which are executed. The manual assembly and calibration actions are not logged and appear as idle time in this log. The wafer scanner is calibrated and tested using calibration and performance software, indicated in the logging as a four-letter code. The logging contains the start and stop moment of each test. The idle time, i.e., the time between stop of the previous test and the start of the next test, is not specified in detail. This idle time has a number of causes, ranging from inexperienced operators reading the procedures, the end of the automated test queue during the night to diagnosing a problem, or waiting for additional parts. Some parts of the test sequence are executed in an automated fashion. The operator starts a test queue which contains a set of test cases which are executed in a sequence. This test queue can also contain part of the recovery and retry sequence for certain failing test cases. The recovery or retry tests are executed depending on the outcome of a test in the queue.

An example fragment of the test log of one of the wafer scanners is depicted in Figure 3(a). Each line corresponds to the execution of one test. The number at the beginning of the line identifies the machine (i.e., the wafer scanner) that is

tested. Afterwards the start time, the completion time, and the four-letter code for the executed test are recorded.[4]

```
1596,31-01-2006 17:33:13,31-01-2006 17:33:39,POLA
1596,31-01-2006 17:33:50,31-01-2006 17:34:46,OSWL
1596,31-01-2006 17:34:48,31-01-2006 17:35:10,OSSP
1596,31-01-2006 17:36:18,31-01-2006 17:36:49,AHZI
1596,31-01-2006 17:42:18,31-01-2006 17:43:25,DSNA
1596,31-01-2006 17:43:39,31-01-2006 17:44:56,AHZI
1596,31-01-2006 17:44:57,31-01-2006 17:59:10,SVEI
1596,01-02-2006 07:15:37,01-02-2006 07:33:25,SVEI
1596,01-02-2006 07:35:00,01-02-2006 07:53:24,SCEI
1596,01-02-2006 07:53:25,01-02-2006 07:54:58,YHLH
1596,01-02-2006 07:54:59,01-02-2006 07:57:41,AHHJ
1596,01-02-2006 07:57:42,01-02-2006 08:04:40,AHCA
```

(a) Fragment of the original log data. Each line corresponds to a test execution with start and end time

```
<ProcessInstance id="1596" description="Test instance 1596">
  ...
  <AuditTrailEntry>
    <WorkflowModelElement>OSWL</WorkflowModelElement>
    <EventType>start</EventType>
    <Timestamp>2006-01-31T17:33:50.000+01:00</Timestamp>
    <Originator>unknown</Originator>
  </AuditTrailEntry>
  <AuditTrailEntry>
    <WorkflowModelElement>OSWL</WorkflowModelElement>
    <EventType>complete</EventType>
    <Timestamp>2006-01-31T17:34:46.000+01:00</Timestamp>
    <Originator>unknown</Originator>
  </AuditTrailEntry>
  ...
</ProcessInstance>
```

(b) Log fragment in MXML format. A separate audit trail entry is created for the start and the end of each test

Fig. 3. Converting the log into the MXML format

To analyze the log data with the ProM framework [18], we had to convert them into the common MXML[5] format. This was performed by means of a custom-built converter plug-in for the ProM*import* framework [21], which facilitates log transformation tasks. In the MXML format, a log is composed of process instances (i.e., cases) and within each instance there are audit trail entries (i.e., events) with various attributes. These attributes refer to, for example, data fields, timestamps, or transactional information (i.e., whether the activity was scheduled, started, or completed). Depending on the kind of information that is in the log, we may be able to answer different questions about the process. Figure 3(b) depicts the MXML log fragment for the highlighted test from Figure 3(a). One can see that the start and the completion of the test are captured by separate audit trail entries (including the corresponding timestamps), and that the enclosing process instance (i.e., the case) corresponds to the tested machine. Because in the original log there were neither data attributes (such as the outcome of a certain test) nor information about originators (such as the people performing the tests) recorded, we will focus on the *control-flow* and *performance* dimension.

Note that the logging takes place on the test-code level, and that there is no reference to the job step in which's context the test is performed. However, in addition to the log data and the job step reference sequence (i.e., a high-level process model describing test dependencies as illustrated in Figure 2), ASML also provided us with an additional document specifying which test codes should be executed in which job step. In this mapping, there are a number of tests that

[4] Note that both the actual machine numbers and the four-letter test codes have been anonymized for confidentiality reasons.

[5] Both the corresponding schema definition and the ProM*import* framework [21], which converts logs from a wide variety of systems to the XML format used by ProM, can be downloaded from *www.processmining.org*.

appear in more than one job step (i.e., are executed in different phases of the test process).

4 Process Mining Results

After converting the log data of the wafer scanners to MXML, we can start with the process mining analysis of the test process execution logs. In the remainder of this section we show to which degree we can already address the questions posed in Section 1 with existing process mining techniques in the ProM framework [18].

First, we look at the log *inspection* facilities that can be used to obtain general information, e.g., about activity distributions, test process durations etc., and apply a number of *filtering* strategies to prepare the data for further processing (Section 4.1). Next, process *discovery* algorithms are applied to mine a model of the test process as it really takes place (Section 4.2). Then, we use *conformance* checking to evaluate how good the test process "matches" both the reference model and the mined models, and apply *extension* techniques for further performance analysis (Section 4.3). Finally, the results are evaluated from an ASML perspective and concrete improvement actions are proposed in Section 5.

4.1 Inspection and Filtering

The ProM framework contains a number of *inspection* tools that provide an overview of the log data. For example, the `Log Summary` provides general information, such as how many cases are captured by the log, how many log events occurred, how many *different* log events (and how often each of them) occurred, which log events happened at the very beginning and which at the very end of a case etc. Figure 4 depicts a screenshot of the log summary for the whole log.

If we examine the log summary, it becomes clear that this test process has very different characteristics compared to typical business processes or administrative processes. In most domains, we typically see a large number of relatively short log traces, i.e., many process instances (cases) each having just a few events. For example, when looking at processes related to patient flows, insurance claims, traffic fines, etc., then there are typically thousands of cases each containing less than 50 events. In this case study, there are just a few cases (i.e., machines) but for each machine there may be thousands of log events. In the initial data set we faced process instances that contained more than 50000 log events (each indicating either the start or the completion of a specific test). As mentioned earlier, the test process of a wafer scanner lasts for several weeks and is partly repeated after the machine has been re-assembled at the customer, thus explaining the huge number of events per machine. From a larger set of machines we selected 24 machines that fulfilled our criteria: (1) the test process needed to be completed, (2) only include the test period on the ASML (and not the customer) site, (3) belong to the same family (recall that typically not more

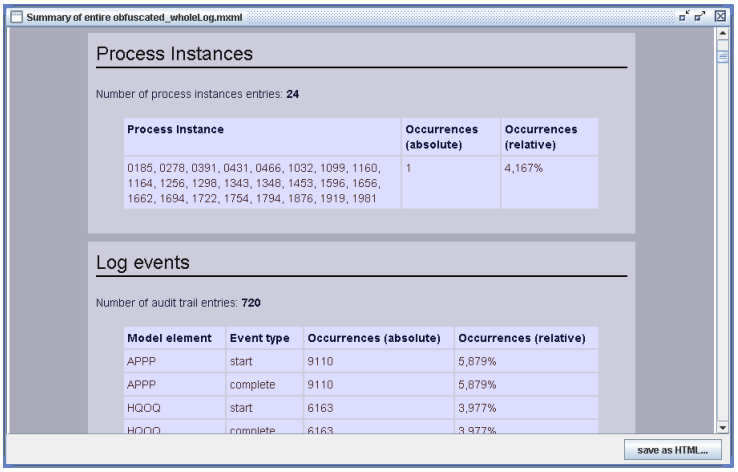


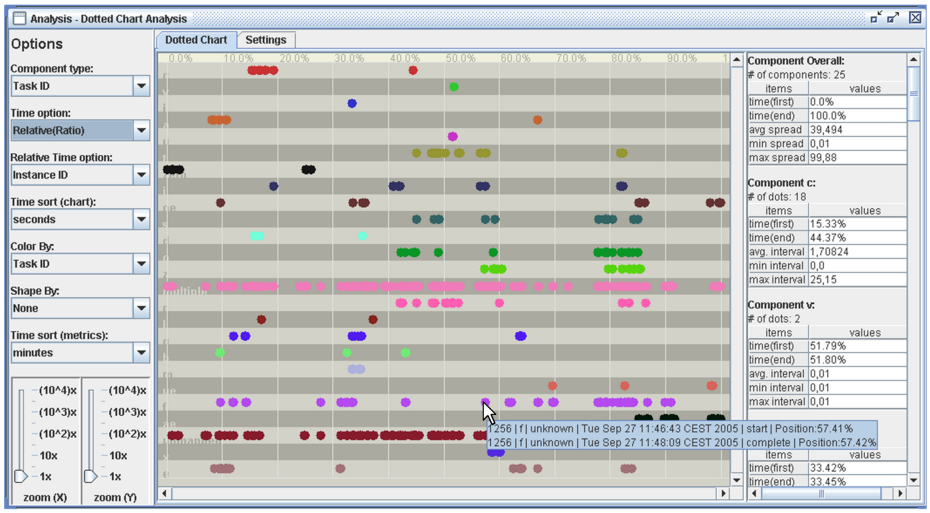
Fig. 4. Screenshot of the Log Summary in ProM. It indicates that the log contains 24 process instances (i.e., tested machines), and that 360 different tests were performed on these machines. The test code that appears most frequently is 'APPP', which is a test that in fact facilitates the execution of another test after some time interval

than 50 wafer scanners of the same type are produced), and (4) not be a pilot system (as a pilot system is used for development testing and not for manufacturing qualification). These 24 cases comprise 154966 log events in total, and the number of log events per process instance (i.e., the length of the test sequence) ranges from 2820 until 16250. Finally, we can see that there are 720 different audit trail entries in the log, which corresponds to 360 different four-letter test codes as each test is captured by both a 'start' and 'complete' event.

As a next step of log inspection, we can make use of the *timestamps* in the log to view the distribution of events over time with the `Dotted Chart Analysis` plug-in. Figure 5(a) depicts a screenshot of this plug-in, where each row corresponds to one of the 24 process instances displayed on a time scale. Note that the displayed time scale does not reveal actual time information for confidentiality reasons, but only shows the logical ordering of events over time. However, different time options (also showing the "real" time) are available. One can see that some of the test processes were long finished when the snapshot was taken, and that systems are being tested in parallel.

The kind of information to be dispersed over the rows can be configured by the user. For example, we can see the activity of different people over time if information about *originators* (i.e., performers) is available in the log. In the case of our test process, we are interested in analyzing the job steps, i.e., the test phases that can be associated to the reference sequence. In Figure 5(b) each

(a) The Dotted Chart analysis visualizes the occurrence of log events over time. Here, the different rows correspond to different process instances, i.e., tested machines. One can see that some of the test processes were long finished when the log snapshot was taken, and that systems are tested in parallel



(b) The entities to be compared over time can be configured. Here, the different rows correspond to different job steps, i.e., phases in the test process, for machine 1256, indicated by the mouse pointer in (a). One can see that some tests are only performed at the beginning or at the end, while others are performed throughout the whole process (such as job step 'f', which is indicated by the mouse pointer)

Fig. 5. Dotted Chart Analysis in the ProM framework

row corresponds to a different job step[6] in the test sequence of one specific machine. Here, it becomes visible that some of the tests are only performed at the beginning or at the end, while others are performed throughout the whole process. This way, frequently reoccurring test phases can be detected and further analyzed.

But to be able to analyze the log on the job-step level, we first have to apply certain *filtering* techniques. Recall that there is no information about job steps recorded in the log, but that we have obtained a document specifying which tests need to be executed for each job step. In this mapping, there are 184 out of the 360 detected test codes associated to a job step. This means that 176 of the four-letter codes cannot be connected to a specific job step (in the remainder of this paper we call them "unmapped" codes). They mainly correspond to additional (more specific) tests that are executed as part of the diagnosis process after a failure. At the same time, there are 49 out of the 184 mapped test codes that are associated to more than one job step, i.e., they occur in different phases of the test process (in the remainder we call them "multiple" codes). The rest of the four-letter codes (i.e., 135 test codes) can be unambiguously mapped onto a specific job step.

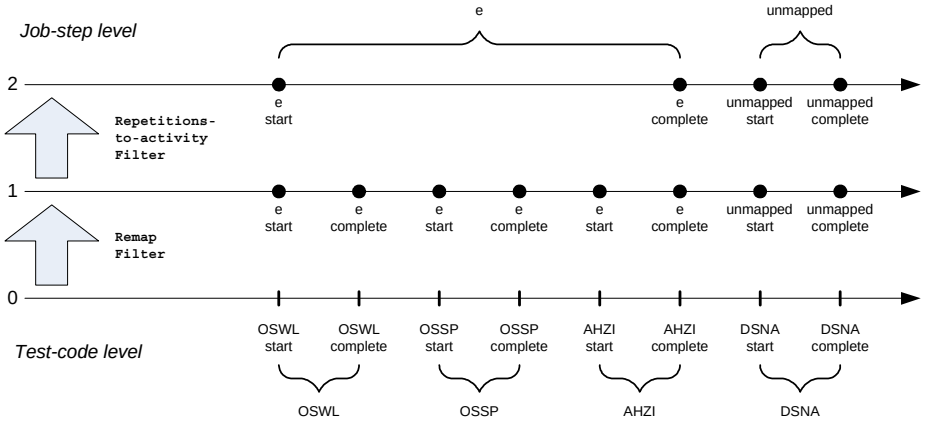


Fig. 6. A combination of filtering techniques was applied to bring the log data from the test-code level to the job-step level

In the ProM framework, there are various *log filter* plug-ins available. They, for example, remove certain audit trail entries from the log. Some log filters can be customized by the user, and each of them (or a combination) can be stored and reused later on. In Figure 6 we show as an example how a part of the log fragment

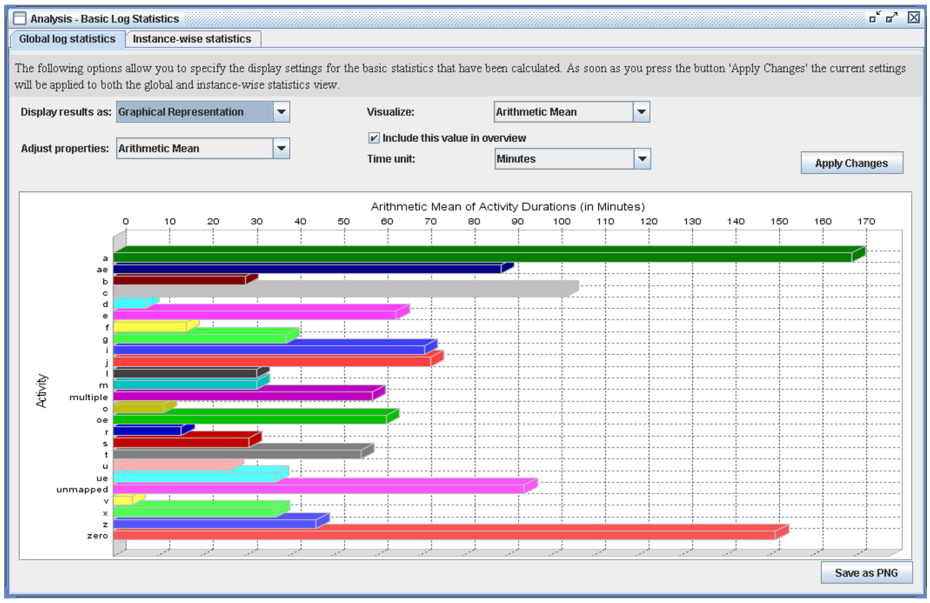
[6] Note that, again, the actual job step names have been replaced by simple letter codes for confidentiality reasons.

from Figure 3 is transferred to the job-step level using a combination of multiple log filters. As a first step, a custom `Remap Filter` is applied. The remap log filter allows for the specification of powerful mappings based on regular expressions. If the name of a log event matches a certain pattern, it can be replaced by another pattern, or the corresponding audit trail entry can be completely removed from the log. Using this filter, we mapped each of the unambiguous test codes onto their corresponding job step identifier, or the 'multiple' or 'unmapped' category if this was not possible. For example, Figure 6 shows that the tests 'OSWL', 'OSSP', and 'AHZI' are associated to the job step 'e', while the test 'DSNA' cannot be mapped to any job step (i.e., 'unmapped'). As a next step, we abstracted from all events that occurred between the first and the last event belonging to the same job step in a row using the `Repetitions-to-activity Filter`. For example, in Figure 6 only the beginning of the first occurrence of a test in job step 'e' (i.e., test 'OSWL') and the end of the last occurrence in job step 'e' (i.e., test 'AHZI') is retained. Note that using this mapping, now also idle times within one job step are covered by the overall job-step duration (for example, the idle time between the completion of test 'OSWL' and the start of test 'OSSP'). As a result, only changes between job steps become visible in the log, which we will use for process discovery on the job-step level later in this paper. For the analysis of job step-related test occurrences over time in Figure 5(b) we used the event log on level 1 (cf. Figure 6), i.e., after the application of the `Remap Filter`.

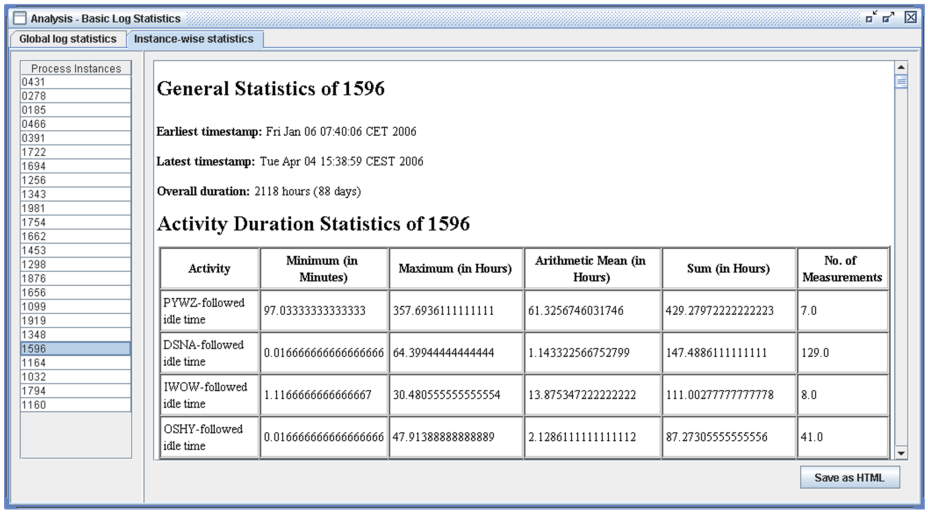
Another way to make use of the timestamp information in the log is to inspect the overall test process durations (i.e., throughput times), and the activity durations and idle times of the test process using the `Basic Log Statistics` plug-in. The plug-in evaluates the 'start' and 'completion' times for each activity in the process. This way, steps in the process that consume much time can be detected and further analyzed. Figure 7(a) depicts a screenshot of the graphical view on the mean time that is spent within each job step. The results are based on the whole log mapped to the job-step level (cf. level 2 in Figure 6). One can see that, on average, most time is spent within job step 'a'.[7]

While each of the statistical measures, such as mean values, variance and standard deviation, can be visualized in a graphical way, the plug-in also offers a textual overview about the results. The measures to be included in the overview can be configured by the user, and the table can be sorted by the names of the activities or any of the available result types. For example, Figure 7(b) shows the textual overview about idle times for the test process of the machine 1596, sorted by the sum of all measured values. One can see that most of the idle time accumulated after the execution of test 'PYWZ'. This can be explained by the fact that this test is used to stabilize the wafer scanner, which takes many hours. Therefore, this test is typically started at the beginning of the weekend, and the

[7] We must keep in mind that a considerable amount of tests could not be uniquely associated (cf. the "unmapped" and "multiple" test categories), and that they are here interpreted as a change of job step, which is not necessarily the case. However, because of the nature of the mapping this is unavoidable.



(a) Graphical representation of the mean time that is spent within one job step for the whole log (includes idle times within the same job step). Most of the time is - on average - spent within job step 'a'



(b) Textual overview about the idle times for one specific process instance (sorted by the sum of all measured values). Most of the idle time accumulated after the test 'PYWZ'

Fig. 7. Basic statistics about activity durations, such as maximum and arithmetic mean values, can be viewed for both single process instances and the whole log

wafer scanner enters idle time if stabilization finishes earlier.

Detailed information about idle times, such as in Figure 7(b), is essential for ASML. As already mentioned in Section 3.2, the goal is to minimize the overall test duration, and, therefore, to decrease the time-to-market. However, before we can extract this information from the log, we again need to apply filtering mechanisms to the initial event log. Figure 8 visualizes how the example log fragment is transformed from representing test durations to idle times after these tests. As a first step, the `Activity-inversion Filter` replaces each 'complete' event and its succeeding 'start' event by a 'start' and 'complete' event marking the transition between those tests.[8] Then, a custom `Remap Filter` is used to abstract from the second element in the transition relation, i.e., the test that was executed afterwards. This is the log that was used as input in Figure 7(b). Nevertheless, it can also be useful to analyze the log on level 1 (i.e., directly after the application of the Activity-inversion Filter) as the succeeding test code can provide insight into the nature of the idle time, e.g., whether the re-execution of parts of the test sequence was necessary, or whether simply the end of an automatic test queue was reached at the end of a night.

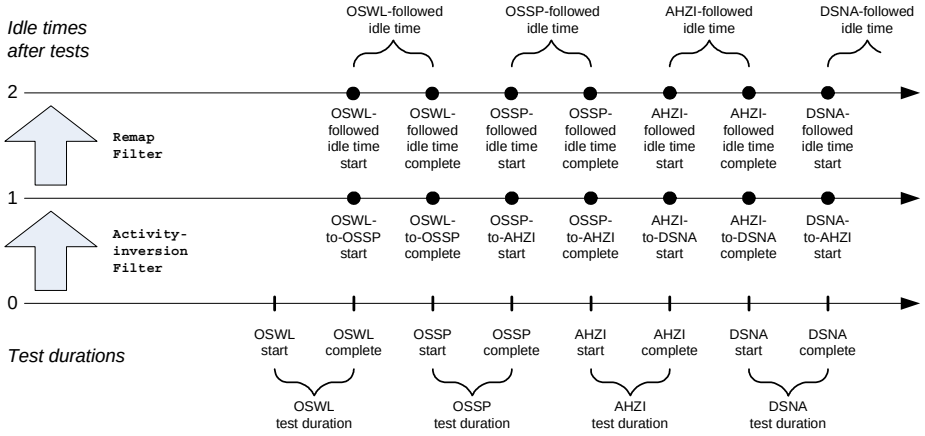


Fig. 8. An inversion filter allows to analyze the idle times instead of the actual test durations. The remap filter is used to abstract from the succeeding test

We have seen that using log inspection and filtering tools we can already answer questions about general log characteristics, such as simple frequency measures, the distribution of events over time, throughput times, and basic statistics about test durations and idle times. In the remainder of this section, we will fo-

[8] Note that this mechanism only works because we have no interleaving tests in the log. Each test that is started will first be completed before the next one is started.

cus on the *control-flow*, i.e., the causal dependencies between steps in the test process. For this, further filtering techniques were applied:

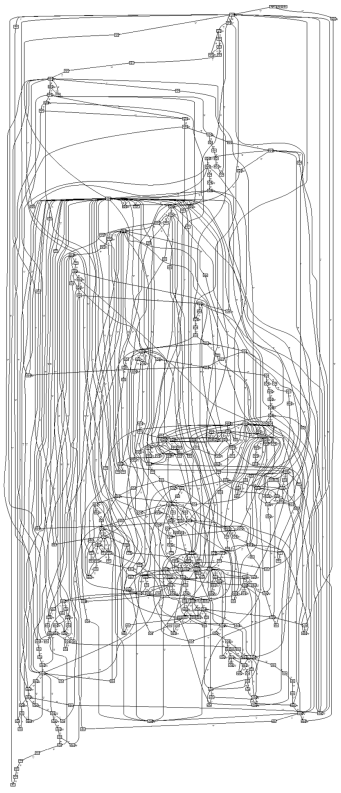
- *Atomic tests.* Often, we can abstract from activity durations if we analyze the control-flow of the process. Therefore, all 'start' events were removed from the log and only the 'complete' events are kept, representing the occurrence of a test in an atomic way.
- *Common behavior.* Only tests that were performed for all the 24 machines were selected using the `Enhanced Event Log Filter`. This filter allows to remove tests that occur below or above a custom commonality (i.e., in how many cases) or frequency (i.e., how often in total) threshold.
- *Mapped test codes.* Only tests that were contained in our mapping document were selected using the simple `Event Log Filter`.
- *Unique test codes.* Only unique mapped codes were selected, i.e., those that occur only in the context of one single job step.
- *Job step executions.* To analyze the changes between the job steps, we built on the filtering depicted in Figure 6, but subsequently abstracted from the 'unmapped' and 'multiple' categories using the `Event Log Filter`. Afterwards, repetitions of the same job step were removed using the `Duplicate Task Filter`.

4.2 Process Discovery

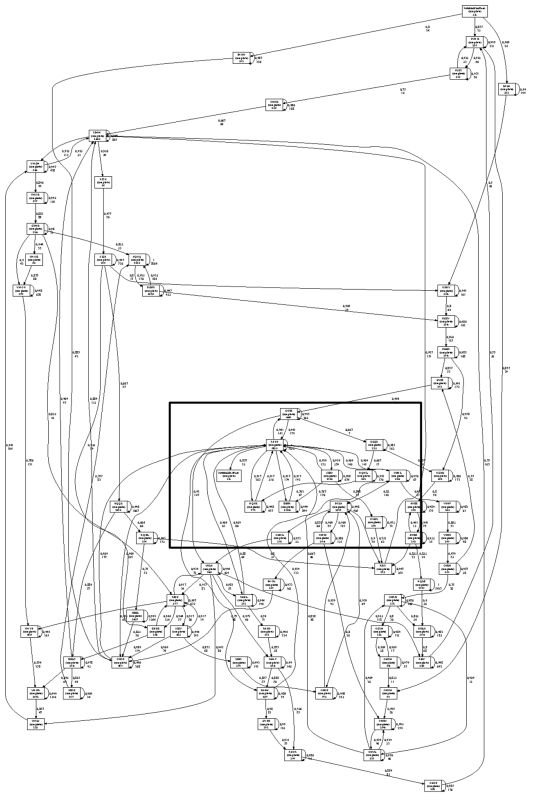
As stated in Section 3.2, the second goal for ASML—next to the minimization of idle times in the test process—is to reduce the re-execution of parts of the job-step sequence. In this section, we want to apply process *discovery* techniques to gain insight into the actual flow of the test process to find out where re-executions were often necessary.

Process discovery algorithms automatically construct a process model based on the behavior that was observed in the event log, and in the ProM framework there are many different discovery plug-ins available. However, the nature of our test log poses some challenges. On the one hand, there are only a few process instances available, which at the same time are very long, and contain logged tests that are executed in different phases of the test process (cf. 'multiple' codes in Section 4.1). On the other hand, we already know that the process is very flexible (as parts of it might need to be redone depending on the outcome of the performed tests), while most of the traditional discovery algorithms described in literature assume that the underlying process is "structured" (i.e., the number of possible paths through a process is limited or the paths have some regular form). Most process discovery plug-ins in ProM also focus on structured processes. Fortunately, there are also several plug-ins that are able to deal with less structured processes [2, 31].

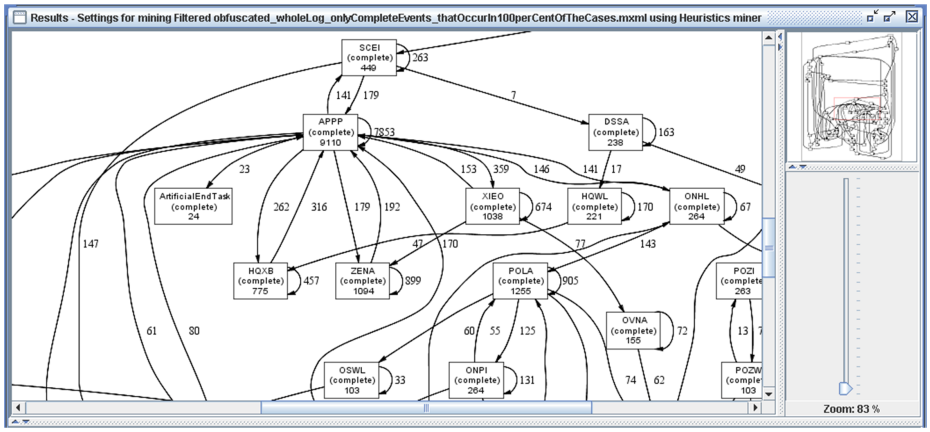
The `Heuristic Miner` is an example of a plug-in that can deal with such less structured processes [31]. It tries to abstract from low-frequent behavior based on certain heuristics to connect the activities in the process. Figure 9(a) depicts



(a) Discovered Process Model based on log with only ‘complete’ events. Contains 360 activities

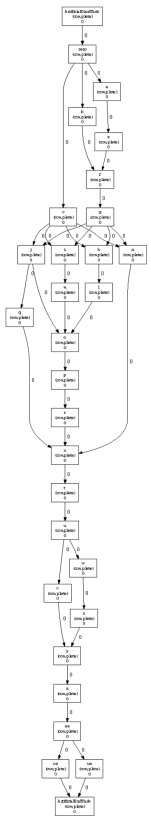


(b) Discovered Process Model based on log with only ‘complete’ events that occurred in all the 24 cases. Contains 70 activities

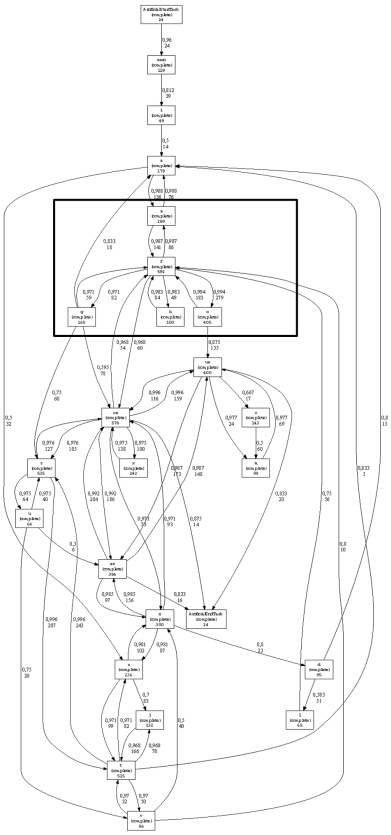


(c) Screenshot of the Heuristic Miner result in ProM. The displayed model part corresponds to the framed area in (b). Boxes correspond to test activities, the numbers within the boxes show how often this test occurred, and the numbers next to the arcs indicate how strong the connection is

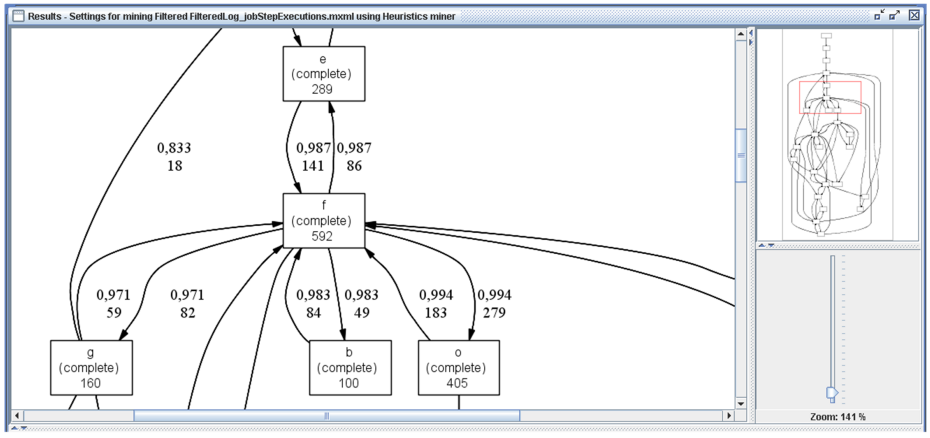
Fig. 9. Process models on test-code level discovery using ProM



(a) Translated reference model for the test process on job-step level



(b) Discovered process model based on log which was mapped onto the job-step level



(c) Screenshot of the Heuristic Miner result in ProM. The displayed model part corresponds to the framed area in (b). In the discovered model the repetitive nature of the test process becomes visible

Fig. 10. Reference sequence and discovered process model on job-step level

the initial model that was discovered based on the whole log (as already indicated, from now on we abstract from the 'start' events and consider activities to be atomic). The discovered model is "spaghetti-like", i.e., it is huge (nodes corresponding to the 360 tests) and it is unstructured. Such spaghetti-like models are not caused by limitations of the plug-in but by the inherent complexity of the testing process. Therefore, we subsequently applied further filtering techniques (cf. list at the end of Section 4.1) to abstract from certain tests in the process. This helps to yield smaller models. As an example, we show the model in Figure 9(b). This model contains only 70 different activities and reflects the view on all *common* tests in the process (i.e., they were performed for each of the machines). Figure 9(c) depicts a screenshot of the mining result in the ProM framework, whereas the displayed area of the process model is highlighted by the rectangle in Figure 9(b). In the modeling formalism used by the `Heuristic Miner`, which is called Heuristics net, boxes correspond to activities (the numbers within the boxes indicate how often this test occurred) and they are connected by directed arcs, whereas the lower numbers next to these arcs indicate how often the connection was observed in the log. One can, for example, see that some tests are frequently repeated (indicated by the directed arc from a test activity to itself), which can be explained by the automated test queues that restart tests after adjusting certain parameters of the wafer scanner as described in Section 3.2. The processes shown in Figure 9 can also be automatically mapped onto more conventional languages such as Event-Driven Process Chains (EPCs), Petri nets, YAWL, etc.

While it is interesting to visualize dependencies on the test-code level, we also want to analyze the process on the job-step level to compare the discovered model to the existing reference sequence. The translated reference sequence is depicted in Figure 10(a), and it reflects the normal flow of the test sequence if nothing goes wrong (i.e., if no test fails). We already know that in reality parts of the test sequence need to be repeated in certain occasions. This also becomes visible in the discovered model based on the log filtered for *job step executions* (cf. list at the end of Section 4.1), which is depicted in Figure 10(b). Note that the discovered process model allows for considerably more paths than the reference model. Figure 10(c), again, shows a screenshot of a part of the mining result in more detail, where one can easily recognize the repetitive nature of the *real* (as opposed to the ideal, i.e., reference) test process. Note that the numbers next to the arcs show the importance of the different paths (the lower number indicates how often this connection was observed in the log, while the upper number indicates the heuristic strength of the corresponding connection). What we can see in Figure 10(c) is a highly connected group around job step 'f', which is bi-directionally connected to the job steps 'e', 'b', 'g', and 'o'. A reason for this effect can be that many executions of the job steps 'e', 'b', 'g', and 'o' result in a re-execution of job step 'f'.

4.3 Conformance and Extension

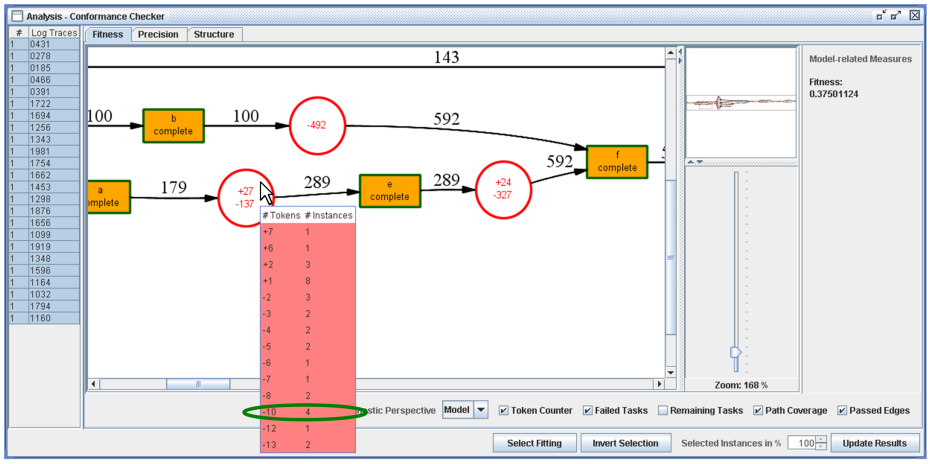
So far, we have seen that it is possible to automatically discover models which represent the behavior that was observed in the event log. But how well is the actual process represented by these models? And to which extent does the observed process comply with the behavior specified in the reference model? Where in the process do most of the deviations occur? These are questions that are addressed by *conformance* techniques, such as the `Conformance Checker` plug-in in the ProM framework. This plug-in measures and visualizes the discrepancies between an event log and a given process model. Another conformance technique is the `LTL Checker` [1], which verifies the compliance of an event log not with respect to a complete process model, but a set of requirements (e.g., business rules).

In the following we show in more detail how the `Conformance Checker` can be used to analyze the conformance of both the reference model and the discovered model on the job-step level (cf. Figure 10) with respect to our test log. As a preparation step, we need to translate the process models from a Heuristics net, as discovered by the `Heuristic Miner`, to a Petri net, which is the modeling language that is supported by the `Conformance Checker`. This can be achieved with the help of one of the various *conversion* plug-ins in the ProM framework. ProM supports different process modeling techniques, e.g., EPCs (the notation used by systems such as SAP and ARIS), different classes of Petri nets, Heuristics nets, different workflow languages (e.g., YAWL and BPEL), etc. These models can be imported, exported, and in most cases it is possible to convert one type of model to another type. The conversion from Heuristics nets to Petri nets is just one of many conversion plug-ins.

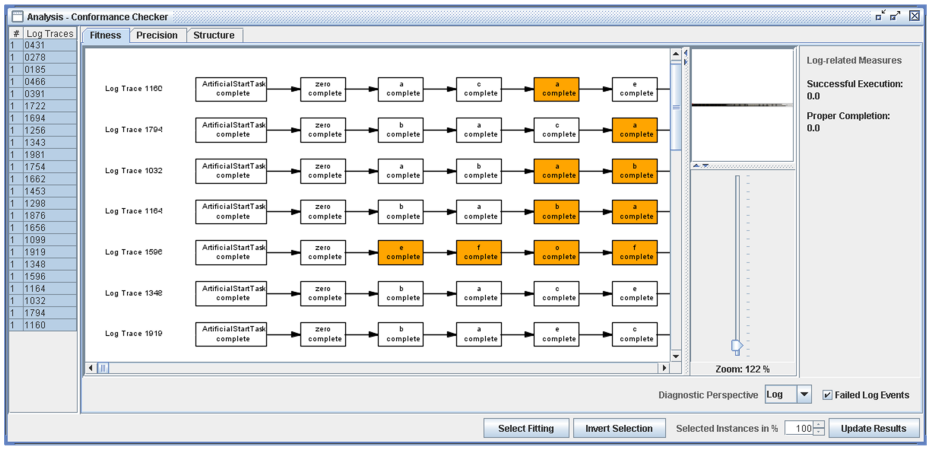
Figure 11(a) depicts a screenshot of the `Conformance Checker`, where the discrepancies between the reference model and the event log (filtered for *job step executions* as described earlier) are visualized from the model perspective. A *log replay*[9] is performed, i.e., each case in the log is "parsed" by the model to analyze the match. During this log replay, tokens may be missing (indicated by a $-$ sign), which indicates that the corresponding task was not ready to be executed according to the model at the time it was performed. For example, in Figure 11(a) one row in the diagnostic feedback for the place before job step 'e' is highlighted. It means that in 4 out of the 24 cases ($\#Instances = 4$) job step 'e' was executed 10 times when it was not possible according to the reference sequence ($\#Tokens = -10$). Such a perceived conformance problem may point us to a repeated execution of a part of the test process (due to a failing test later in the sequence). Similarly, remaining tokens (indicated by a $+$ sign) hint that the following activity was expected to be executed according to the model (but in fact was not).

The log view of the `Conformance Checker` helps to further narrow down potential root causes of the discrepancy. Figure 11(b) shows a screenshot of the log view (highlighting those log events that do not comply with the process

[9] The interested reader is kindly referred to [28] for further details about the described conformance checking techniques.



(a) Here, the model view shows how many tokens remained or were missing in the place before job step 'e', i.e., 'e' was expected to be executed according to the reference model (but in reality was not), or 'e' was not ready to be executed according to the model (but in reality still was)



(b) The log view visualizes for each process instance which log events could be successfully replayed in the reference model (i.e., the corresponding activity was ready to be executed) and which did not match (highlighted in orange color)

Fig. 11. The Conformance Checker measures and visualizes discrepancies between the log and a given process model

description). One can see that in the top-most log trace the first mismatch occurs after the initial 'zero' step and the executions of job step 'a' and 'c'. In the log we then find a second execution of job step 'a', which in not allowed according to the model. However, job step 'a' is not causally related to job step 'c', and as described in Section 3.2, it is in fact allowed to change between such parallel job steps within the test process, which is not captured by the process model. This shows that not every detected conformance problem necessarily corresponds to a re-execution of a part of the test sequence. Domain knowledge is needed to investigate the discrepancies between the actual test process and the reference model in more detail, and we will report on the evaluation of our findings from an ASML perspective in Section 5.

However, the `Conformance Checker` also measures the degree of *fitness* based on the amount of missing and remaining tokens during log replay [28], i.e., it quantifies to which degree the log traces comply with a given process model. This fitness analysis clearly indicates that the discovered model is much more representative for the observed test process than the reference model (cf. fitness values in Table 1).

Table 1 contains the fitness values for each of the test instances with respect to both the reference sequence and the discovered model on the job-step level as depicted in Figure 10, whereas possible values range from 0.0 (corresponds to the case where the model and the log do not fit at all) to 1.0 (i.e., model and log fit to 100%). Furthermore, it shows how many job step executions were contained in the filtered log for each machine (column before the last column in Table 1), and how many test code events were originally recorded for this machine (last column in Table 1). Finally, in the bottom row average values are given for all the 24 machines. We can see that, although the discovered process model does not completely "match" the behavior observed in the log, it clearly fits much better than the reference sequence. This is not surprising as we already know that—in contrast to the discovered model—the reference model does not capture the possible repetitions in the process at all, but it describes the ideal flow of the process if nothing goes wrong. So, the discovered model is a much better representation of the test process as it took place, which demonstrates that process mining can provide insight into how processes are *really* executed.

Finally, if we have a good process model (which may be handcrafted or obtained through process discovery), *extension* techniques can be used to project information which was extracted from the log onto the existing model. For example, performance bottlenecks can be visualized directly in the process model. Figure 12 depicts a screenshot of the `Performance Analysis with Petri net` plug-in in ProM, which graphically highlights performance-related information, such as the waiting time between two activities, directly in the given process specification. According to the severity, places are colored in red (high), yellow (medium), or blue (low waiting time). However, for this kind of analysis the plug-in assumes that the model and the log are perfectly fitting each other. This is not the case for any of our models of the test process, and therefore we can only use it to display performance information with respect to activities, such

Table 1. Fitness values indicating the degree of compliance for each of the test instances with respect to both the reference sequence and a discovered process model. Clearly, the discovered model fits much better than the overall model

Machine ID	*Fitness Reference Process Model*	*Fitness Discovered Process Model*	*No. of Filtered Job-step Events*	*No. of Original Test-code Events*
0431	$f = 0.309$	$f = 0.751$	238	6504
0278	$f = 0.385$	$f = 0.828$	270	6136
0185	$f = 0.376$	$f = 0.717$	206	5710
0466	$f = 0.356$	$f = 0.745$	422	8162
0391	$f = 0.384$	$f = 0.727$	159	3902
1722	$f = 0.334$	$f = 0.760$	301	6270
1694	$f = 0.397$	$f = 0.782$	526	10408
1256	$f = 0.410$	$f = 0.744$	222	5722
1343	$f = 0.399$	$f = 0.701$	130	5360
1981	$f = 0.357$	$f = 0.667$	551	12670
1754	$f = 0.402$	$f = 0.776$	192	16250
1662	$f = 0.414$	$f = 0.769$	182	3830
1453	$f = 0.405$	$f = 0.596$	164	6410
1298	$f = 0.378$	$f = 0.424$	170	3852
1876	$f = 0.356$	$f = 0.753$	150	4538
1656	$f = 0.368$	$f = 0.656$	126	2820
1099	$f = 0.424$	$f = 0.672$	193	3946
1919	$f = 0.337$	$f = 0.727$	205	5048
1348	$f = 0.410$	$f = 0.638$	184	5240
1596	$f = 0.410$	$f = 0.581$	224	5784
1164	$f = 0.376$	$f = 0.672$	499	10860
1032	$f = 0.324$	$f = 0.706$	301	6896
1794	$f = 0.394$	$f = 0.734$	114	2972
1160	$f = 0.405$	$f = 0.770$	186	5676
Average	$f = 0.375$	$f = 0.711$	246.458	6456.917

as the time which is spent between two user-defined activities. In Figure 12, the job steps ‘a’ and ‘e’ are selected, and the average, minimum, etc. time between them is displayed.

5 Evaluation and Improvement Suggestions

To identify concrete improvement suggestions, we evaluated the presented process mining results from an ASML perspective. In a first analysis effort, we mainly investigated the differences between the reference model and the discovered model. First, the order of job steps was analyzed (Section 5.1). The job-step order is the sequence in which job steps are executed in the factory. Some variation is allowed, but not too much. We investigated whether—according to the discovered model as in Figure 10—the test process followed the reference process (including the allowed variations). Second, the dominant feedback loops in the test process were analyzed (Section 5.2). Feedback loops in the process indicate

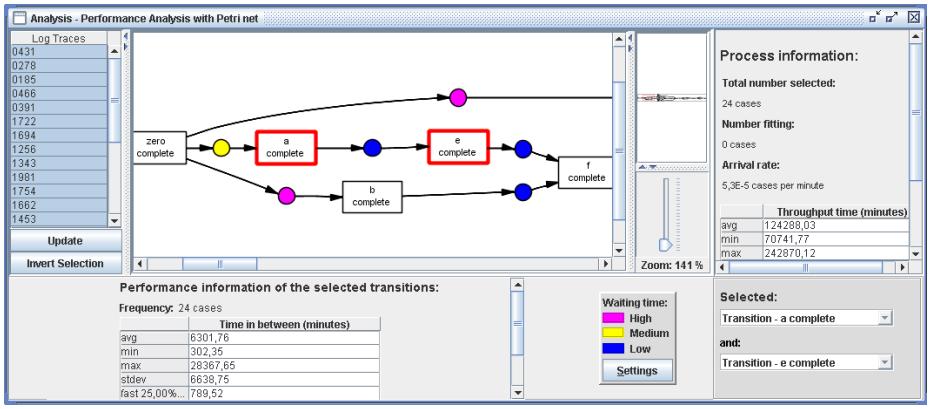


Fig. 12. The Performance Analysis with Petri-net plug-in graphically projects time information (such as waiting time between two activities) onto a given process model

that a job step fails and a previous job step needs to be re-done. After the previous job step is re-done, often parts of the already executed sequence must also be re-executed. This is taking valuable time. Finally, possible root causes for some of the idle times are given (Section 5.3). The reduction of idle times can help to decrease the overall test process duration, and, therefore, decrease the time-to-market.

5.1 Order of Job Steps in the Test Process

When we investigated whether the real process followed the reference process, considering the allowed variations, we obtained three types of results: (1) job steps that are actually executed on a different place in the reference sequence (i.e., deviations from the process model shown in Figure 10(a)), (2) groups of highly connected job steps, and (3) job steps that are not in the reference sequence but in the test log. In the following, we describe them in more detail.

1. It appeared that job step 'i' was positioned in 81% of the cases just after the 'zero' job step, i.e., at the beginning of the discovered process model, while—according to the reference sequence—it should be executed in the middle of the test process. While looking for possible root causes for this difference, we realized that a newer version of the reference sequence was released in the end of 2006. The main change in the new reference sequence was that job step 'i' and 'j' were positioned just after the 'zero' job step at the beginning of the test sequence. The analyzed systems were build up according to the new sequence for job step 'i'. Interestingly, job step 'j' was still found in the original position. If job step 'j' is really to be executed in the beginning of the sequence, then active steering should take place to align the test execution. Note that we also re-checked the conformance of the test

log with respect to the updated reference sequence, but the fitness values did not change significantly (on average $f \approx 0.45$).

2. Two highly connected groups of job steps are discovered in the discovered process model. The first group is depicted in Figure 10(c), a strong connection between job step 'f' and a number of other job steps: 'e', 'b', 'g' and 'o'. These connections are bi-directional between 'f' and the other job steps. A reason for this effect could be that any execution of the job steps 'e', 'b', 'g' and 'o' results in a re-execution of job step 'f'. Job step 'f' is a relatively short job step which can be executed automatically. As a result, the entire test set is executed. Specific (parts) of the test set in job step 'f' could be faster when job step 'f' needs to be executed after job step 'e', 'b', 'g' and 'o' are executed. In general, speeding up job step 'f' is beneficial because it is executed multiple times in the entire sequence.

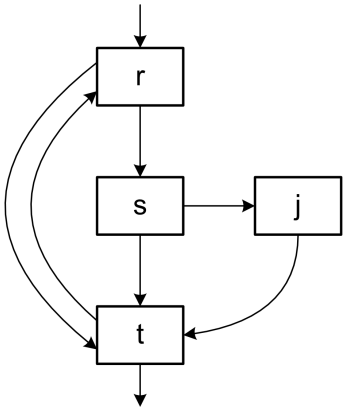


Fig. 13. Highly connected group of job steps, which has been identified based on the process mining results presented in Section 4

The second highly connected group is centered around job steps 'r', 's', 't' and 'j'. The mined process showed the following pattern (see Figure 13). Job step 'r' and 't' are bi-directionally connected. Job steps 'r', 'j' and 't' are illumination steps, while job step 's' is a non-illumination step. The root cause of a failure of job step 't' is solved by job step 'r'. A re-execution of job step 'r' causes a re-execution of job step 's' (and possibly 'j'). An improvement proposal would be to introduce a more thorough test in job step 'r' (i.e., add a similar test to the one in job step 't') which causes that, if the failure occurs, it already occurs in job step 'r' and can be immediately fixed in job step 'r'. This prevents the re-execution of job step 's' (and possibly 'j').

3. One of the feedback loops revealed that job step 'd' is executed, although it is not in the reference sequence. Job step 'd' is currently not investigated to be improved to decrease the cycletime, because this job step is not supposed to

be executed. The process mining results revealed that job step 'd' is executed as part of a recovery plan. Job step 'd' could be further investigated for cycle time reduction.

5.2 Dominant Feedback Loops

Feedback loops in the mined process indicate that a certain job step failed and caused that a job step, which was positioned earlier in the sequence, needs to be re-executed. For example: job step 10 fails – a repair takes place (hardware, software or machine parameters change) – job step 5 must be re-executed to calibrate the hardware/software/machine parameters – job steps 6 to 10 need to be re-executed as well.

Ideally, failures, fixes, and re-execution of test cases are performed in the job step that failed. Given technological constraints related to the construction of a wafer scanner, this is not always feasible. Moreover, many different faults can cause a particular failure. Process improvement in this area should start with the most important failures and feedback loops. The following dominant feedback loops (→) have been observed in the discovered process model: (1) z → (via d) → a, (2) z → (via d) → l, (3) t → a, and (4) v → f.

In general, this shows that the job steps 'z', 't', and 'v' are job steps which are capable of finding the errors that were missed in the previous job steps. Errors detected in these steps cause the test process to be "rolled back". Test cases that fail in these job steps should be placed in the lower level job steps as additional test case to test the overall performance. This allows an early failure and an early fix. This might be the reason that job step 'i' and 'j' (both illumination job steps) were positioned earlier in the updated reference sequence (cf. Section 5.1). Process improvement should focus on these four dominant feedback loops for these systems. More specifically, the first and second feedback loop visit job step 'd'. This is possibly to determine if either job step 'a' or job step 'l' needs to be executed to fix this problem. The test cases in job step 'd' that perform this diagnosis could be useful as a standard test case in job step 'a' and 'l'.

5.3 Idle Times during Test Procedure

Using idle time for scheduling automatic test actions is one of the possibilities to reduce the overall test duration. In Section 4.1 we described how the idle time accumulating after a certain type of test can be determined using a combination of filtering techniques and the application of the `Basic Log Statistics` plug-in. Figure 7(b) depicts the analysis results for one of the 24 machines. Most of the idle time on this machine accumulated after executing the tests (1) 'PYWZ', (2) 'DSNA', and (3) 'IWOW', which are also among the 4 tests that accumulated most of the idle time for all the 24 machines together.

(1) The 'PYWZ' test is used to stabilize the wafer scanner, which takes hours. Therefore, this test is started at the beginning of the weekend at the end of the 'zero' job step. The wafer scanner enters idle time if stabilization finishes earlier. This can easily be solved by adding the test cases of the next job step to this

test set. In the case of the machine depicted in Figure 7(b), at least 97 minutes, and on average 61 hours accumulated after 7 executions of this test.

(2) The 'DSNA' test ensures the reliability of the wafer handler, one of the sub-systems in a wafer scanner. This test is executed during the available night hours throughout the entire job step sequence. Sometimes, the weekend is too long for the test queue. This can be resolved by adding additional 'DSNA' test cases to the test set. For the machine in Figure 7(b), at least 0.01 minutes, and on average 1.14 hours accumulated after 129 executions of this test.

(3) After the execution of the test 'IWOW', at least 1 minute, and on average 13 hours accumulated after 8 executions on the same machine. The test was executed twice as the last test before the weekend, and once before the Christmas holiday.

We have demonstrated that current process mining techniques can already answer many questions, even yield concrete suggestions for process improvement also in as complex environments as the wafer stepper qualification phase of ASML. However, due to the rapid technological advancements, the analysis results presented in this paper are likely to be outdated already for the next series of wafer steppers than the ones that we analyzed. To enable a continuous improvement of the test process, process analysis should be best carried out in an iterative manner.

At the same time, we also face interesting challenges posed by the size of the log (initially more than 150,000 log entries) and the few process instances (log data from only 24 tested wafer scanners were available for analysis). Even more challenging is the extremely flexible nature of the process, where people testing these machines may change back and forth between job steps if they do not need to be synchronized (to use the otherwise idle time in the most efficient way), and where whole parts of the process may need to be repeated due to a failing test later in the test sequence. This is the reason why—in contrast to other case studies (e.g., a municipality [16] and the National Public Works Department [3] in the Netherlands)—our current process mining algorithms discover "spaghetti-like" process models. Consider, for example, the discovered process model in Figure 9(a). In fact, there is nothing wrong with discovering such a spaghetti-like model as it correctly shows that the underlying process behavior is very diverse. But it is not very helpful for gaining insight into the main process, distinguishing certain scenarios, and answering specific questions. In such a case, it is crucial to be able to *abstract* from some of the details. In Section 4.1, we achieved different levels of abstraction using various filtering techniques. However, filtering the log as shown requires some work, is not interactive, and we lose the connection to the previous stage of filtering (e.g., we cannot decide to expand one of the job steps to view this part on the test-code level). Therefore, the next section discusses some research challenges related to the mining of processes like the test processes of ASML and other spaghetti-like processes encountered in other domains (e.g., health-care processes).

6 Challenges for Process Mining

In Section 4 it was shown that with existing process mining techniques present in the ProM framework [18], already compelling results can be achieved. Note that the logs considered in this paper are more challenging than the logs of more structured processes (e.g., workflows in administrative organizations). Hence, it is fair to conclude that applying process mining to real-life logs is a viable option. However, sometimes the resulting models are overly complex and confusing (i.e., "spaghetti-like"), which makes them hard to extract useful information from. In this section, we report on two ProM plug-ins whose development was inspired by the case study reported in this paper: the `Cloud Chamber Miner` and the `Frequency Abstraction Miner`. These new plug-ins have been developed with the test processes of ASML in mind.

To motivate the development of the two new plug-ins, we first summarize limitations and assumptions of existing tools. It can be argued that the current practice of process mining is founded on a, partially implicit, set of assumptions which need to be adjusted to succeed in less structured, more flexible situations as found in real life (e.g., ASML's test processes and health-care processes).

- *Noise*, i.e. incorrect or corrupted information in logs, is already expected and dealt with in a number of mining approaches. However, recent analyses of real-life logs have shown that the concrete quantity and quality of noise differs significantly from initial, clean-room assumptions, and also between distinct situations.
- Current algorithms focus strongly on a *precise description* of the observed behavior, preferably using formalized and exact models. In practice, the observed reality is often very complex. In many cases one would thus prefer a *less precise* description, where one can abstract from less relevant details.
- When faced with unknown log data, one needs extensive time to explore the structure and particularities of that specific data set, so that mining tools can be correctly adjusted to the situation. For this task, the complex and inflexible nature of current mining techniques is often a burden. *Interactive tools for data exploration* could dramatically improve the quality of results and reduce the effort necessary, by leveraging on user knowledge.

As introduced in Section 4.1, filtering is a powerful means for improving the specific quality of raw log data. However, the filters currently available are not sufficient to tackle the types of noise found in real-life logs. Thus, in many practical applications *proprietary filters* have to be developed, which are targeted towards a specific (type of) log file. Lessons learned from these custom implementation efforts include:

- Filtering is not limited to deleting events; it is also often necessary to *modify* events or to introduce *new events*.
- More advanced filters need to have a more *global view on the log data*, rather than basing decisions on local events considered in isolation.

– Providing *fine-grained customization options* to the user, while picking *useful default configurations* tremendously increases the efficiency of using a filter.

The development of more effective and versatile log filters is thus a necessity for the successful interpretation of real-life log data. Before log filters can be configured to remove noise from the data, however, one needs to determine what unwanted noise *exactly is* in the first place. This usually involves a back-and-forth cycle between filtering and mining, adjusting filters based on the observation of unwanted artifacts in mining results.

We are currently experimenting with techniques to visualize event log data appropriately for pre-processing tasks. The goal is to find a visualization that enables practitioners to spot data anomalies quickly, and to determine effective means for removing them. One technique for data visualization which relies heavily on the human brain's ability to spot geometric patterns are *dotplots* [23]. The data is laid out on both axes of a raster (from top-left to top-right, and from top-left to bottom-left); for every coordinate where the x and y coordinates represent the same event class, the pixel is painted with a color symbolizing the frequency of the respective event. Otherwise, the pixel is painted black. To be more precise: both x and y range over all possible events, i.e., the concatenation of all events in all cases. A pixel (x, y) is painted black if x and y refer to different events. Only if x and y "match", pixel (x, y) is colored. This way, the diagonal axis from the top-left towards the bottom-right represents the self-similarity of the data (by definition). But every pattern that appears off the axis reflects some kind of similarity pattern (see [23] for more information and practical examples, such as detecting duplicated code in a software system).

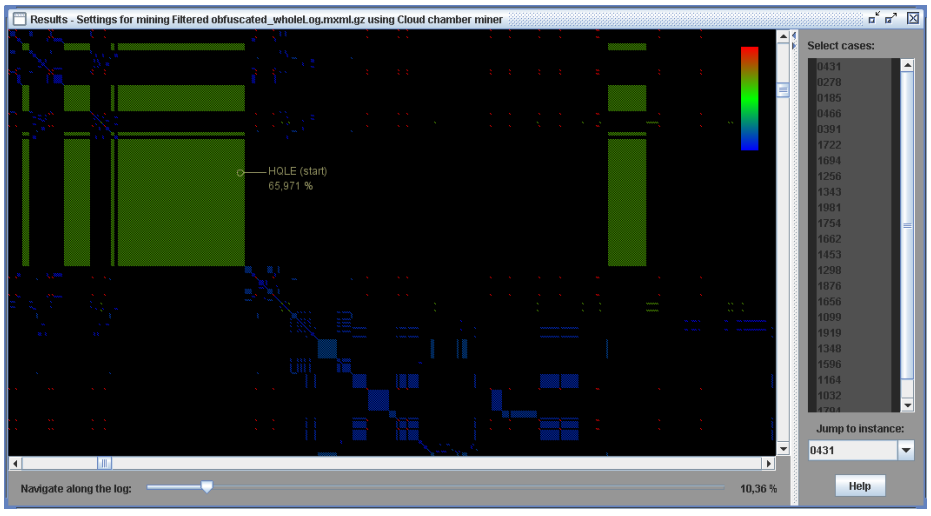


Fig. 14. ProM's Cloud Chamber Miner enables log exploration by dotplot visualization

Figure 14 shows the `Cloud Chamber Miner`, a plug-in recently added to ProM, which uses the idea of dotplots and applies this to process mining. Figure 14 shows only a fraction of the entire dotplot. Both the x and y coordinates refer to all events for all machines (i.e., the whole log). The log's self-similarity can be clearly seen as a diagonal line extending from the upper left corner, i.e., all coordinates (x, x) are colored. Several repetitive patterns are visible in the screenshot in Figure 14. For example, the biggest square around the diagonal axis shows the repeated executions of the 'HQLE' test. The green color indicates that this is a test having a moderate frequency (about 66% relative frequency compared to the most-frequent test in the log). This test is one of the convergence test cases, where the performance of the projection system is measured for a number of typical user settings. The log contains a number of event classes, which are repeated frequently. These short loops, which refer to the automatic re-starts of test steps, show up as square patterns in the dotplot visualization. An interesting feature of dotplots is that it is possible to see patterns when focusing on a single machine but also between different machines, e.g., if the same sequence occurs in multiple machines, then diagonal lines are shown. While the dotplot visualization is still very close to the log, the `Cloud Chamber Miner` provides means for navigating to specific process instances, and for exploring the log in real time. This feature is important, in that it enables and encourages experimentation, which is essential in the process mining domain.

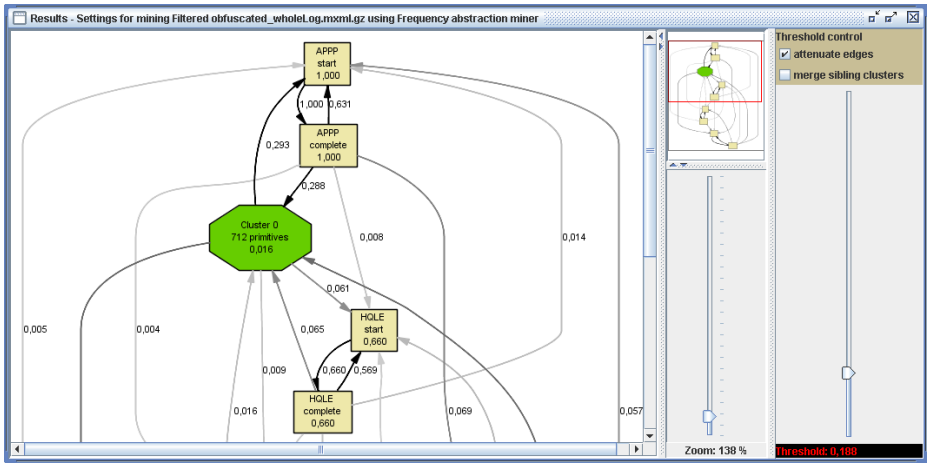


Fig. 15. The Frequency Abstraction Miner in ProM can aggregate low-frequent behavior in clusters

The most challenging task in process mining, however, remains to find suitable techniques for the high-level analysis of process log data. One line of research focuses on making these techniques more flexible, less precise, and more interactive. Figure 15 shows ProM's `Frequency Abstraction Miner`, an exper-

imental technique that attempts to satisfy these requirements. It features one single control, a slider that allows the user to specify a frequency threshold value. Whenever this slider is moved, the plug-in will cluster all events below the frequency threshold. Clusters are visualized as green octagonal nodes, while less frequently observed relations are depicted using a lighter greytone. This behavior significantly simplifies the displayed process model, allowing the user to focus on the most frequent behavior in the log.

In contrast to filtering as described in Section 4.1, the abstraction provided by the `Frequency Abstraction Miner` is both *dynamic* and *non-destructive*, i.e. it can be adjusted interactively and does not modify the original data. Clusters can also be "expanded", i.e. when the user clicks on a cluster node a subgraph will be shown, detailing the primitive nodes the cluster is composed of and their respective relations.

Both the `Cloud Chamber Miner` and the `Frequency Abstraction Miner` represent but first steps on the road to more explorative tools for process mining. Future work in this direction will complement the present set of more precise algorithms, in that it enables users to interactively learn about the log data. Using this knowledge, the data can be filtered appropriately, so that also more precise techniques can be applied while avoiding "spaghetti results".

The development of these techniques is driven and inspired by our experience in practical applications, like the ASML case described in this paper and the analysis of the event logs of several Dutch hospitals. We expect that an improved and extended toolkit will allow us to perform case studies like this in a much shorter timeframe, while providing more insightful results.

7 Conclusion

To conclude the paper, we first reflect on our findings in the case study and then make some more general comments on the applicability of process mining.

Analyzing the event log of a test process of ASML's wafer scanners using process mining techniques revealed potential improvements in the following two areas:

1. *Conformance of the actual executed sequence with the reference sequence.* The mined process revealed that another process was executed than the reference sequence, i.e., the fitness of the reference process compared to reality was only 37.5 percent. An updated version of the reference sequence was released in the middle of the observed period. However, the discovered process model revealed that the new reference sequence was not followed entirely and that there are many deviations.
2. *Concrete improvement proposals for the failures which disturb the test process the most.* Improvement proposals were based on a number of different observations. First, groups of strongly connected job steps were discovered. This resulted in specific suggestions that could make the process more transparent and faster. Second, one job step that was found in the log was not contained in either of the reference sequences. This job step was executed

as part of the diagnosis and analysis feedback loops, and could be further analyzed for optimization. Third, four dominant feedback loops were identified in the discovered model of the test process. Case-by-case analysis is still required to see how these feedback loops can be removed. Finally, possible root causes for some of the idle times were given. Further investigation of idle times can lead to a more efficient usage and reduction of the overall test duration.

The typical characteristics of the test process of an ASML wafer scanner are that a highly variable reference sequence is allowed and a relatively small number of wafer scanners of one system type are made. The application of process mining techniques in ASML should therefore start with the first system of a new wafer scanner type. Analysis and adjustment of the sequence should then be done in a "closed loop" after each new system is tested and delivered. A closed-loop process would look like the following:

1. Determine the reference sequence.
2. Test the wafer scanner.
3. Analyze the log and reference sequence for each tested wafer scanner.
4. Determine the most dominant feedback loops and job step groups.
5. Determine improvement actions for the next untested wafer scanner (more wafer scanners can be in the pipeline already). Typical improvement actions are: (a) moving test cases to other job steps / duplicating test cases, (b) moving test cases from job steps to sub-system level, (c) updating the reference sequence, and (d) the normal actions (parts quality, test process, machine status monitoring, etc. but with focus on the dominant feedback loops).
6. Continue until all systems are delivered.

Additional information sources that could be used to improve the data set are available, but were not used in this case study. SAP data is available with (manually entered) job step start and stop data, and the start and stop moment of the entire test sequence. This data is less reliable, because it is manually entered and it contains job steps that overlap in time. However, together with the other log data, better process models could be discovered. The individual test results (pass, fail, out of specification) could be used to determine why a job step is stopped and a feedback loop is started. Furthermore, the start and stop moments of test queues can be added as information. This information is available in the ASML systems, but needs conversion to the ProM format.

This paper describes the first case study that applies process mining to less structured processes. All *prior applications of process mining reported in literature have been focusing on rather structured administrative processes.* In our case study, we applied our ProM framework to the test process of ASML. As described, the testing of ASML's wafer scanners is a highly dynamic process and a real challenge for existing process mining techniques. Moreover, there are many other domains where similar logs can be obtained (e.g., health-care processes). Therefore, our findings are also relevant to these other domains. The case study

shows that filtering is very important, e.g., for many questions we had to map low level events (test codes) onto aggregate events (job steps). After doing the appropriate filtering we were able to demonstrate the applicability of all three types of process mining: (1) "discovery", (2) "conformance", and (3) "extension" (cf. Figure 1). Using these three types of process mining, we could answer all three questions raised in the introduction: (1) "How are the tests actually executed?", (2) "How compliant are the actual test executions to the reference process?", and (3) "Where is the most time spent in the test process?".

Despite the positive results, the case study also showed that the mining of less structured processes is far from trivial and requires expert knowledge and perseverance. Therefore, using the case study, we discussed limitations of existing algorithms and presented two new process mining techniques inspired by the mining of ASML's test processes. Both techniques have been implemented as plug-ins in ProM and are subject to further research.

Acknowledgements

This research is supported by the Technology Foundation STW, EIT, the EU project SUPER, and the IOP program of the Dutch Ministry of Economic Affairs. Furthermore, the authors would like to thank all ProM developers for their on-going work on process mining techniques.

References

1. 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. In R. Meersman and Z. Tari et al., editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005*, volume 3760 of *Lecture Notes in Computer Science*, pages 130–147. Springer-Verlag, Berlin, 2005.
2. 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 of *Lecture Notes in Computer Science*, pages 48–69. Springer-Verlag, Berlin, 2005.
3. W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713–732, 2007.
4. 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.
5. 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.
6. 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.

7. S. Amland. Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. *Journal of Systems Software*, 53(3):287–295, 2000.
8. ASML. Website ASML NV, Veldhoven, The Netherlands, www.asml.com, 2007.
9. R. Boumen, I.S.M. de Jong, J.W.H. Vermunt, J.M. van de Mortel-Fronczak, and J.E. Rooda. A risk-based stopping criterion for test sequencing. Internal Report SE 420460, Eindhoven University of Technology, January 2006. Submitted to IEEE Transactions on Systems,Man,and Cybernetics-Part A: Systems and Humans.
10. R. Boumen, I.S.M. de Jong, J.W.H. Vermunt, J.M. van de Mortel-Fronczak, and J.E. Rooda. Test sequencing in complex manufacturing systems. *Accepted for IEEE Transactions on Systems,Man,and Cybernetics-Part A: Systems and Humans*, 2006.
11. 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.
12. 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.
13. 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.
14. Chipworks corporation. Advanced semiconductor manufacturing handbook. Technical report, Chipworks corporation, Januari 2000.
15. A. Datta. Automating the Discovery of As-Is Business Process Models: Probabilistic and Algorithmic Approaches. *Information Systems Research*, 9(3):275–301, 1998.
16. A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Department of Technology Management, Technical University Eindhoven, 2006.
17. B. van Dongen and W.M.P. van der Aalst. Multi-Phase Mining: Aggregating Instances Graphs into EPCs and Petri Nets. In D. Marinescu, editor, *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 35–58. Florida International University, Miami, Florida, USA, 2005.
18. B. 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 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
19. 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 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin, 2004.
20. D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M.C. Shan. Business process intelligence. *Computers in Industry*, 53(3):321–343, 2004.
21. C.W. Günther and W.M.P. van der Aalst. A Generic Import Framework for Process Event Logs. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops, Workshop on Business Process Intelligence (BPI 2006)*, volume 4103 of *Lecture Notes in Computer Science*, pages 81–92. Springer-Verlag, Berlin, 2006.
22. M.J. Harrold, D. Rosenblum, G. Rothermel, and E. Weyuker. Empirical studies of a prediction model for regression test selection. *IEEE Transactions on software engineering*, 27(3):248–263, 2001.

23. J. Helfman. Dotplot Patterns: a Literal Look at Pattern Languages. *TAPOS*, 2(1):31–41, 1995.
24. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
25. IDS Scheer. ARIS Process Performance Manager (ARIS PPM): Measure, Analyze and Optimize Your Business Process Performance (whitepaper). IDS Scheer, Saarbruecken, Gemany, http://www.ids-scheer.com, 2002.
26. British Computer Society Special Interest Group in Software Testing. *Standard for Software Component Testing*. British Computer Society, 2001.
27. M. zur Mühlen and M. Rosemann. Workflow-based Process Monitoring and Controlling - Technical and Organizational Issues. In R. Sprague, editor, *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS-33)*, pages 1–10. IEEE Computer Society Press, Los Alamitos, California, 2000.
28. 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 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, Berlin, 2006.
29. M. Sayal, F. Casati, U. Dayal, and M.C. Shan. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.
30. A. Streit, B. Pham, and R. Brown. Visualisation Support for Managing Large Business Process Specifications. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *International Conference on Business Process Management (BPM 2005)*, volume 2678 of *Lecture Notes in Computer Science*, pages 205—219. Springer-Verlag, Berlin, 2005.
31. 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.