Big Software on the Run

In Vivo Software Analytics Based on Process Mining (Keynote)

Wil van der Aalst
Eindhoven University of Technology
P.O. Box 513, 5600 MB
Eindhoven, The Netherlands
w.m.p.v.d.aalst@tue.nl

ABSTRACT

Software-related problems have an incredible impact on society, organizations, and users that increasingly rely on information technology. Specification, verification and testing techniques aim to avoid such problems. However, the growing complexity, scale, and diversity of software complicate matters. Since software is evolving and operates in a changing environment, one cannot anticipate all problems at design-time. Hence, we propose to analyze software "in vivo", i.e., we study systems in their natural habitat rather than through testing or software design. We propose to observe running systems, collect and analyze data on them, generate descriptive models, and use these to respond to failures. We focus on *process mining* as a tool for *in vivo* software analytics. Process discovery techniques can be used to capture the real behavior of software. Conformance checking techniques can be used to spot deviations. The alignment of models and real software behavior can be used to predict problems related to performance or conformance. Recent developments in process mining and instrumentation of software make this possible. This keynote paper provides pointers to process mining literature and introduces the "Big Software on the Run" (*BSR*) research program that just started.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; H.2.8 [**Information Systems**]: Database Applications—*data mining*; D.2.8 [**Software Engineering**]: Metrics—*software science*

General Terms

Measurement, Performance, Verification

Keywords

Process mining, software analytics, event logs, process discovery, conformance checking, software engineering

ICSSP'15 , August 24–26, 2015, Tallinn, Estonia

1. SOFTWARE CHALLENGES

Software forms an integral part of the most complex artifacts built by humans. Software systems may comprise hundreds of millions of program statements, written by thousands of different programmers, spanning several decades. Their complexity surpasses the comprehension abilities of any single, individual human being [3]. Accordingly, we have become totally dependent on complex software artifacts. Communication, production, distribution, healthcare, transportation, education, entertainment, government, and trade all increasingly rely on "Big Software". Unfortunately, we only recognize our dependency on software when it fails. Malfunction information systems of the Dutch police force and the Dutch tax authority, outages of electronic payment and banking systems, increasing downtime of high-tech systems, unusable phones after updates, failing railway systems, and tunnel closures due to software errors illustrate the importance of good software.

Problems are not limited to crashing and incorrectly operating software systems. Software may also allow for security breaches (unauthorized use, information leakage, etc.), cause performance problems (long response times, etc.), or force people to use the system in an unintended manner.

There is no reason to assume that things will get better without a radical change of paradigm. We see the following challenges [3]:

- *Growing complexity*: software systems do not operate in a stand-alone manner, but are increasingly interconnected resulting in complex distributed systems.
- *Growing scale*: increasing numbers of organizations use shared infrastructures (e.g. cloud computing), the number of devices connected to the internet is increasing, and an increasing number of data is recorded (e.g. sensor data, RFID data, etc.).
- *Increasing diversity*: there is a growing diversity in platforms (covering traditional CPUs, multi-core architectures, cloud-based data centers, mobile devices, and the internet of things), versions (different releases having different capabilities), and configurations.
- *Continuous evolution of software*: late composition (components are assembled and connected while they are running), remote software updates (removing old errors but introducing new ones), and functional extensions (by merely changing running software) lead to unpredictable and unforeseen behavior.

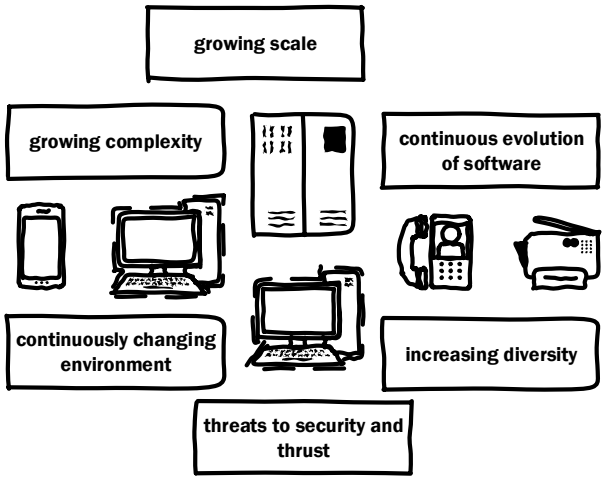

Figure 1: Challenges for today's and tomorrow's software systems.

- *Continuously changing environment*: the software must run in an ever-changing context of (virtualized) hardware, operating systems, network protocols and standards, and must cope with wild variations of available resources, such as computer cores, bandwidth, memory and energy. Moreover, the software may be applied in ways not anticipated at design time.
- *Increasing demands related to security and trust*: as our reliance on software grows, concerns about security and privacy increase.

To date, the computer science discipline has tried to address such problems by proposing new design methodologies and programming/specification languages. However, these *a priori* techniques have inherent limitations: we cannot predict evolving requirements and circumstances at design time and numerous examples show that traditional approaches cannot cope with the complexity of today's information systems. *Despite progress in software engineering, we still lack the knowledge and expertise to build large software systems that are reliable, robust, secure, fast, and well-aligned with continuously changing circumstances.* This is the reason that manufacturers of electronic devices and information systems have started to monitor actual system behavior. They are recording problems (e.g. monitoring system crashes and component failures), but typically only address problems in a trial-and-error fashion.

We propose to more systematically exploit the enormous amounts of event data are already being recorded by/from software systems running in their natural habitat. We accept that software may malfunction and study software systems in their natural environment to better understand the problems and to minimize their impact.

2. BIG SOFTWARE ON THE RUN (BSR)

This paper is inspired by the "Big Software on the Run" (*BSR*) research program [7] that started in 2015. The program will run for a period of four years and is supported by the three Dutch technical universities (Eindhoven University of Technology, TU Delft, and University of Twente). It was initiated by 3TU.NIRICT, the Netherlands Institute for Research on ICT, which comprises all ICT research of the three universities of technology in the Netherlands. The program is based on an earlier national grant proposal involving additional partners such as Vrije Universiteit (VU) and Radboud University Nijmegen (RUN) [3].

The *BSR* research program proposes to *shift the main focus from a priori software design to a posteriori software analytics thereby exploiting the large amounts of event data generated by today's systems.* The core idea is to study software systems *in vivo*, i.e., at runtime and in their natural habitat. We would like to understand the *actual* (desired or undesired) behavior of software. Running software needs to adapt to evolving and diverging environments and requirements. This forces us to consider software artifacts as "living organisms operating in changing ecosystem". This paradigm shift requires new forms of empirical investigation that go far beyond the common practice of collecting error messages and providing software updates.

Unlike traditional testing approaches (both white- and black-box testing), we focus on analyzing the software system in its natural environment rather than a controlled experiment conducted offline. Moreover, we consider many variants of the same software system running under possibly very different circumstances. Most testing approaches are not applied in vivo and try to cripple the system using usual and unusual input. We study systems in their natural habitat and try to avoid that systems crash or are slowed down. Unlike research on fault tolerant design and runtime verification, we focus on understanding the software system first. *We do not assume that we can always define the correct behavior and corresponding countermeasures upfront. Instead, we aim to learn as much as possible from the information derived from run time data, and subsequently we use this knowledge to diagnose problems and recommend actions.* Breakthroughs in discovery, conformance checking, and prediction are tightly coupled to novel interactive visualizations.

Figure 2 shows an overview of the *BSR* research program. Running software systems are the result of software development processes that we would like to provide with results from in vivo software analytics. To do this we need to extract events from the running software. This can be done in two ways: (1) the code is instrumented with logging functionality and (2) the data sets managed by the software are extracted and transformed to event data. In both cases, event data revealing the actual software behavior are created as input for analysis. Part of the *BSR* research program focuses on *process mining* [1] as a tool to analyse behavior based on such event data. The following techniques are used:

- *Discovery techniques* aim to discover process models from event data and provide insightful visualizations of the actual software behavior. The results reveal how software systems are actually being used.
- *Conformance checking techniques* aim to detect deviations from normative or descriptive models. The model may be based on domain knowledge or based on discovered models.
- *Prediction techniques* aim to predict functional and non-functional properties of running cases, individual systems, and classes of systems.

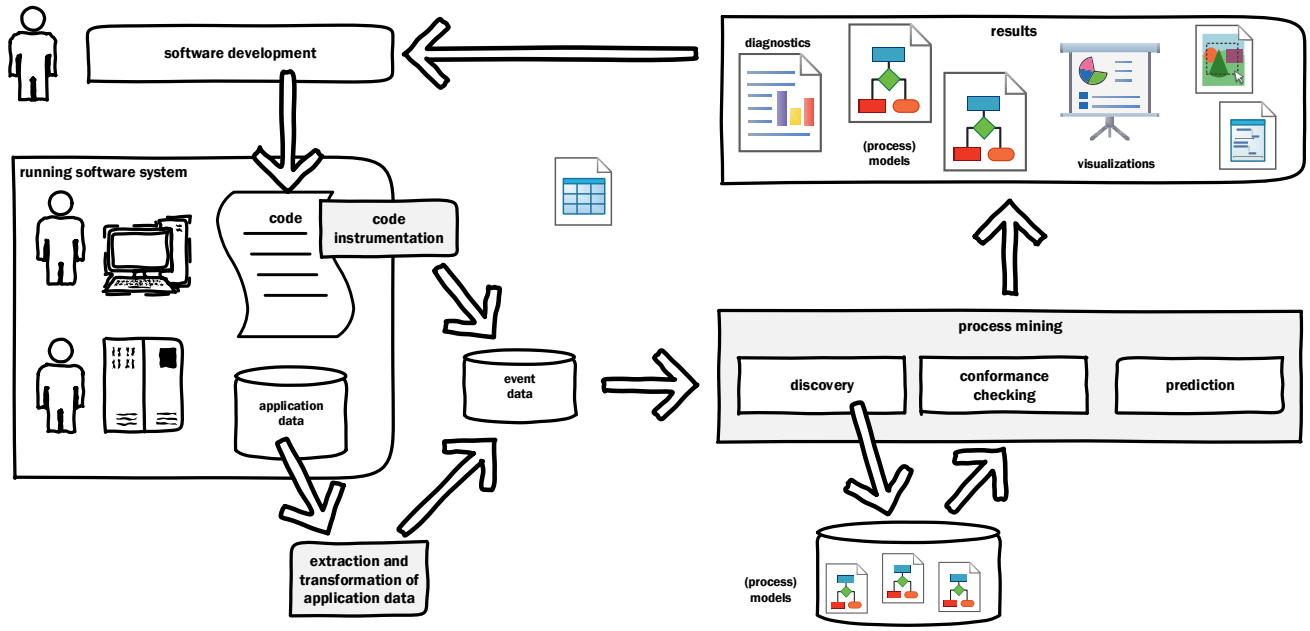

Figure 2: Overview of the "Big Software on the Run" (*BSR*) research program and the role of process mining in the *BSR* program.

As shown in Figure 2, diagnostics, (process) models, and a wide variety of visualizations are provided as input to the software development process.

Since the focus is on the *behavior* software systems and not on their structure, process models play a central role in the *BSR* research program. Note that process models serve as both input and output for analysis (see Figure 2).

3. SOFTWARE AS A LIVING ORGANISM

To explain our focus on *in vivo software analytics*, it is helpful to view a software system as a *living organism* (like a human being). Whereas traditional approaches in computer science aim to create new "perfectly designed organisms" from scratch, we try to better understand the relationship between characteristics of individuals, lifestyles, symptoms, and health-related problems and use this information to recommend corrective measures. Instead of designing new design methodologies and programming/specification languages, we want to provide a comprehensive set of tools for the diagnosis, treatment, and prevention of software-related problems.

Collecting event data can be seen as recording blood pressures, taking X-rays, examining blood counts, etc. *Medical symptoms may point to diseases, just like system crashes, service calls, and user complaints indicate software-related problems.* We would like to identify patterns that indicate a particular type of software problem. To understand a disease and its causes, one needs to monitor many patients. To understand software-related problems (i.e., diseases), we need to compare event data from many different sources. The same software or different versions/variants of the same software may be used by different organizations and users. In a similar vein to everyday health problems, software problems may be inherited (e.g. built upon instable software components or platforms), accidental (e.g. unanticipated integration problems) or due to lifestyle factors (e.g. unintended use or overloading). Fortunately, modern computing infrastructures allow for the collection of event data from different sources and use these data for comparative analysis.

Just like humans, software systems need to adapt to changing circumstances. However, unlike diseases or traumas, software problems are *not* caused by wear, heavy use, or aging. Key problems are continuously changing functionality and circumstances as well as mankind's limited ability to predict such changes. An additional problem is that software systems do not operate in isolation. To truly understand the behavior and usage of software, one needs to consider the entire ecosystem consisting of a variety of interconnected software systems. Our focus on in vivo software analytics makes it possible to study complex software ecosystems. Our analysis will not be restricted to individual software components. This is reflected by our desire to study "Big Software" systems in their natural habitat and to emphasize the interaction and interference between different subsystems.

We hope to provide a *classification of software-related problems*, similar to the ICD classification in healthcare. The International Classification of Diseases (ICD) provides codes to classify diseases and a wide variety of signs, symptoms, abnormal findings, complaints, and external causes of injury or disease. The ICD is published by the World Health Organization (WHO) and is used for morbidity and mortality statistics, reimbursement systems, and automated decision support in health care. We would like to understand software at a comparable level. Unfortunately, *the diversity in software systems is enormous* compared to humans. Humans –despite being unique– have many common features (e.g. two legs, ten fingers, one liver, and one heart),

whereas software systems tend to be much more diverse. Moreover, we would like to understand evolving software ecosystems consisting of a variety of interconnected subsystems. Our classification will not focus on software-related problems only. Software has the amazing ability to function properly for decades when left alone. There are software artifacts that can adapt to frequent changes and remain operational for an extended period. Therefore, we would like to learn from these best practices.

4. PROCESS MINING AS AN ENABLER

The paper emphasizes the role of process mining in the *BSR* research program. As shown in Figure 2, we consider three main types of process mining: *discovery*, *conformance checking*, and *prediction*.

Input for process discovery is an *event log*. Each *event* in such a log refers to an *activity* (i.e., a well-defined step in some process) and is related to a particular *case* (i.e., a *process instance*). The events are partially ordered. The set of events related to a case describes one "run" of the process. (Hence, the term process instance.) Such a run is often referred to as a *trace*. It is important to note that an event log contains only example behavior. In the software domain, it is particularly challenging to choose a suitable case notion [17]: What are the process instances in the software system studied? The choice determines the scope and nature of the process models discovered or used.

Process discovery, i.e., discovering a process model from a multiset of example traces, is a very challenging problem and various discovery techniques have been proposed [5, 6, 8, 9, 11, 12, 13, 15, 18, 19, 23, 27, 29, 30]. Many of these techniques use Petri nets during the discovery process. It is impossible to provide a complete overview of all techniques here. Very different approaches are used, e.g., heuristics [13, 29], inductive logic programming [15], state-based regions [5, 12, 27], language-based regions [9, 30], and genetic algorithms [23]. Inductive mining techniques based on so-called process trees provide various guarantees while still being able to analyse large and noisy event logs [18, 19, 20].

Conformance checking aims to "confront" process models (discovered or modeled by hand) with real-behavior as recorded in event logs. Various techniques have been developed [2, 14, 22, 24]. The more advanced conformance checking techniques create *alignments*, i.e., observed behavior is related to modeled behavior even if there are deviations.

Prediction techniques use a combination of historic data, learned models, and the current state of the software system. It may be possible to learn so-called "problem signatures", i.e., patterns in event data that point to particular problems [10]. See [21] for a generic prediction framework.

See [25, 26] for recent case studies applying process mining to software systems.

5. CONCLUSION

This paper proposes to analyze software systems under real-life circumstances using process mining. This approach is followed in the "Big Software on the Run" (*BSR*) research program [7] and pointers to existing process mining approaches are provided in this paper.

However, the application of process mining to running software is still in its infancy. Lion's share of process mining literature focuses on the operational processes supported by the software rather than the software itself.

Future work will focus on better instrumenting software systems. Source code transformation, binary weaving, or techniques to capture communication events are used to create the events. In [17] the jointpoint-pointcut approach from Aspect-Oriented Programming (AOP) is used to instrument the software. Apart from technical challenges, we also face the more conceptual challenge to select a proper case notion. In [17] process instances are based on user requests (called business transactions). However, different case notions are possible.

We will also try to exploit the specifics of software. The fact that there is an architecture (implicit or explicit), can be exploited by process mining techniques. For example, in [4] it is show that the "location" of the event in some software architecture can be used to discover better process models. Events often refer to a software component, class, service, or some other entity. This provides insights into (im)possible or (un)desirable interactions between parts of the software system. We can define patterns and anti-patterns [16, 28] that can be checked against the *actual* software behavior.

6. ACKNOWLEDGMENTS

The author would like to thank Arie van Deursen, Jack van Wijk, Inald Lagendijk, Jaco van de Pol, Marieke Huisman, Henri Bal, Erik Poll, Bart Jacobs, Maikel Leemans, Boudewijn van Dongen for their participation in the original Big Software on the Run (*BSR*) research proposal or/and the current *3TU.BSR* research program (see `www.3tu-bsr.nl`).

7. REFERENCES

[1] W. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.

[2] W. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

[3] W. van der Aalst, H.Bal, A. van Deursen, B. Jacobs, I. Lagendijk, and J. van Wijk. BSR: Big Software on the Run. Gravitation Research Proposal, 2013.

[4] W. van der Aalst, A. Kalenkova, V. Rubin, and E. Verbeek. Process Discovery Using Localized Events. In R. Devillers and A. Valmari, editors, *Applications and Theory of Petri Nets 2015*, volume 9115 of *Lecture Notes in Computer Science*, pages 287–308. Springer-Verlag, Berlin, 2015.

[5] W. van der Aalst, V. Rubin, H. Verbeek, B. van Dongen, E. Kindler, and C. Günther. Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010.

[6] W. van der Aalst, A. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[7] W. van der Aalst, J. van Wijk, A. van Deursen, I. Lagendijk, J. van de Pol, and M. Huisman. 3TU.BSR: Big Software on the Run. `www.3tu-bsr.nl`, 2015.

[8] R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer-Verlag, Berlin, 1998.

[9] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007.

[10] J.C. Bose and W. van der Aalst. Discovering Signature Patterns from Event Logs. In B. Hammer, Z. Zhou, L. Wang, and N. Chawla, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2013)*, pages 111–118, Singapore, 2013. IEEE.

[11] J. Carmona and J. Cortadella. Process Mining Meets Abstract Interpretation. In J. Balcazar, editor, *ECML/PKDD 210*, volume 6321 of *Lecture Notes in Artificial Intelligence*, pages 184–199. Springer-Verlag, Berlin, 2010.

[12] J. Carmona, J. Cortadella, and M. Kishinevsky. A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In *Business Process Management (BPM2008)*, pages 358–373, 2008.

[13] J. Cook and A. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

[14] D. Fahland, M. de Leoni, B. Dongen, and W. van der Aalst. Behavioral Conformance of Artifact-Centric Process Models. In A. Abramowicz, editor, *Business Information Systems (BIS 2011)*, volume 87 of *Lecture Notes in Business Information Processing*, pages 37–49. Springer-Verlag, Berlin, 2011.

[15] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust Process Discovery with Artificial Negative Events. *Journal of Machine Learning Research*, 10:1305–1340, 2009.

[16] G. Hohpe and B. Woolf. *Enterprise Integration Patterns*. Addison-Wesley Professional, Reading, MA, 2003.

[17] M. Leemans and W. van der Aalst. Discovering Real-Life Business Transactions and Process Models from Distributed Systems. (technical note), 2015.

[18] S. Leemans, D. Fahland, and W. van der Aalst. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In N. Lohmann, M. Song, and P. Wohed, editors, *Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2013)*, volume 171 of *Lecture Notes in Business Information Processing*, pages 66–78. Springer-Verlag, Berlin, 2014.

[19] S. Leemans, D. Fahland, and W. van der Aalst. Discovering Block-structured Process Models from Incomplete Event Logs. In G. Ciardo and E. Kindler, editors, *Applications and Theory of Petri Nets 2014*, volume 8489 of *Lecture Notes in Computer Science*, pages 91–110. Springer-Verlag, Berlin, 2014.

[20] S. Leemans, D. Fahland, and W. van der Aalst. Exploring Processes and Deviations. In F. Fournier and J. Mendling, editors, *Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2014)*, volume 202 of *Lecture Notes in Business Information Processing*, pages 304–316. Springer-Verlag, Berlin, 2015.

[21] M. de Leoni, W. van der Aalst, and M. Dees. A General Framework for Correlating Business Process Characteristics. In S. Sadiq, P. Soffer, and H. Voelzer, editors, *International Conference on Business Process Management (BPM 2014)*, volume 8659 of *Lecture Notes in Computer Science*, pages 250–266. Springer-Verlag, Berlin, 2014.

[22] M. de Leoni, W. van der Aalst, and B. van Dongen. Data- and Resource-Aware Conformance Checking of Business Processes. In W. Abramowicz, D. Kriksciuniene, and V. Sakalauskas, editors, *Business Information Systems (BIS 2012)*, volume 117 of *Lecture Notes in Business Information Processing*, pages 48–59. Springer-Verlag, Berlin, 2012.

[23] A. Medeiros, A. Weijters, and W. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.

[24] A. Rozinat and W. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.

[25] V. Rubin, I. Lomazova, and W. van der Aalst. Agile Development with Software Process Mining. In *Proceedings of the 2014 International Conference on Software and System Process (ICSSP 2014)*, pages 70–74. ACM Press, New York, NY, USA, 2014.

[26] V. Rubin, A. Mitsyuk, I. Lomazova, and W. van der Aalst. Process Mining Can Be Applied to Software Too! In M. Morisio, editor, *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14)*, pages 57:1–57:8, New York, NY, USA, 2014. ACM.

[27] M. Sole and J. Carmona. Process Mining from a Basis of Regions. In J. Lilius and W. Penczek, editors, *Applications and Theory of Petri Nets 2010*, volume 6128 of *Lecture Notes in Computer Science*, pages 226–245. Springer-Verlag, Berlin, 2010.

[28] N. Trcka, W. van der Aalst, and N. Sidorova. Data-Flow Anti-Patterns: Discovering Data-Flow Errors in Workflows. In P. van Eck, J. Gordijn, and R. Wieringa, editors, *Advanced Information Systems Engineering, Proceedings of the 21st International Conference on Advanced Information Systems Engineering (CAiSE'09)*, volume 5565 of *Lecture Notes in Computer Science*, pages 425–439. Springer-Verlag, Berlin, 2009.

[29] A. Weijters and W. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

[30] J. van der Werf, B. van Dongen, C. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 94:387–412, 2010.