Software Trajectory Analysis: An empirically based method for automated software process discovery

Pavel Senin
Collaborative Software Development Laboratory
University of Hawaii at Manoa
POST 307, 1680 East-West Rd.
Honolulu, HI 96822
senin@hawaii.edu

ABSTRACT

A process defines a set of routines which allow to organize, manage and improve activities to reach a goal. With an expert intuition and apriori knowledge software processes were modeled for the long time resulting in Waterfall, Spiral and other development models. Later, with a wide use of SCM systems and public availability of primitive software process artifact trails, formal methods such as Petri Nets, State Machines and others were applied for recurrent processes discovery and control. Recent advances and increase in use of continous integration and in-process software measurement and analysis systems rich types of software process artifacts becoming available. In this work I propose to investigate an automated technique for the discovery and characterization of recurrent behaviors in software development - “programming habits” either on an individual or a team level.

1. INTRODUCTION AND MOTIVATION

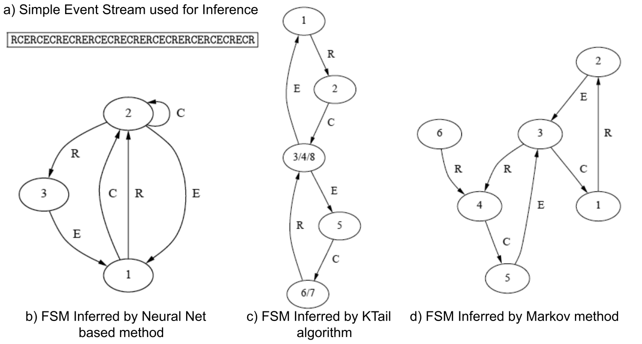
A *software process* is a set of activities performed in order to design, develop and maintain software systems. Examples of such activities include design methods; requirements collection and creation of UML diagrams; testing and performance analysis. The intent behind a software process is to structure and coordinate human activities to achieve the goal - deliver a software system successfully; while the objective of software process research is to design a process consisting of a set of recurrent behaviors which allows to reproduce such a success consistenly and improve the process performance.

Much work has been done in software process research resulting in a number of industrial standards for process models (CMM, ISO, PSP etc. [4]) which are widely accepted by many government and industrial institutions. Nevertheless, software development remains error-prone and more than a half of all software development projects ending up failing or being very poorly executed. Some of them are abandoned due to running over budget, some are delivered with such low quality or so late that they are useless, and some, when delivered, are never used because they do not fulfill requirements [12]. The cost of this lost effort is enormous and may in part be due to our incomplete understanding of software process.

There is a long history of software process improvement through proposing specific patterns of software development. For example, the Waterfall Model process proposes a sequential pattern in which developers first create a Requirements document, then create a Design, then create an Implementation, and finally develop a Tests. On the contrary, the Test Driven Development process proposes an iterative pattern in which the developer must first write a test case, then write the code to implement that test case, then refactor the system for maximum clarity and minimal code duplication. Probably the main problem with this traditional top-down approach to process development is that it requires the developer or manager to notice a recurrent pattern of behavior in the first place [4].

As an alternative to that, in my research, I am applying knowledge discovery and data mining techniques to the domain of software engineering in order to evaluate their ability to automatically notice interesting recurrent patterns of behavior from collected software process artifacts. While I am not proposing to be able to infer a complete and correct software process model, my system will provide its users with a formal description of recurrent behaviors in their software development. As a simple example, consider a development team in which committing code to a repository triggers a build of the system. Sometimes the build passes, and sometimes the build fails. To improve the productivity of the team, it would be useful to be aware of any recurrent behaviors of the developers. My system might generate one recurrent pattern consisting of a) implementing code b) running unit tests, c) committing code and d) a passed build: $i \rightarrow u \rightarrow c \rightarrow s$, and another recurrent pattern consisting of a) implementing code, b) committing code, and c) a failed build: $i \rightarrow c \rightarrow f$. The automated generation of these recurrent patterns can provide actionable knowledge to developers; in this case, the insight that running test cases prior to committing code reduces the frequency of build failures.

2. RELEVANT PRIOR WORK


a) Simple Event Stream used for Inference
RCERCECRECRERCECRECRERCECRERCERCECRECR
1
R
E
2
C
3/4/8
E
R
5
C
6/7
2
C
R
3
C
R
E
E
1
2
E
6
3
R
R
C
R
4
E
1
C
5

b) FSM Inferred by Neural Net based method

c) FSM Inferred by KTail algorithm

d) FSM Inferred by Markov method

Figure 1: Process discovery through the grammar inference: panel a) a sample event stream (simple process involving three types of events: Edit, Review, and Checkin); and FNA results obtained by applying three methods of process discovery from Cook & Wolf [7].

Although process mining in the business domain is a well-established field with much software developed up to date (ERP, WFM and other systems), "Business Process Intelligence" tools usually do not perform process discovery and typically offer relatively simple analyzes that depend upon a correct a-priori process model [22] [2]. This fact restricts direct application of business domain process mining techniques to software engineering, where processes are usually performed concurrently by many agents, are more complex and typically have a higher level of noise. Taking this fact in account, I will review only the approaches to the process mining for which applicability to software process was expressed.

Perhaps, the research most relevant to my own was done by Cook & Wolf in [7]. The authors developed a *"process discovery"* techniques intended to discover process models from event streams. The authors did not attempt to generate a complete model, but rather to generate sub-models that express the most frequent patterns in the event stream. They designed a framework which collects process data from history logs, and generates a set of recurring patterns of behavior characterizing observed process. In this work they extended two methods of *grammar inference* from previous work: purely statistical (neural network based *RNet*) and purely algorithmic (*KTail*) as well as developing their own Markovian method (*Markov*).

Process discovery, in the author's opinion, resembles the process of grammar inference, which can be defined as a process of inferring a language grammar from the given set (sample) of sentences in this language. In the demonstrated approach, words of the language are atomic events of the dynamic process, whether sentences built from such words, are describing the behavior of a process. Consequently, the inferred grammar of that language is the formal model of the process. Cook & Wolf expressed such grammars as Finite State Machines (FSMs) and implemented a software tool for the mining of the software process. This tool was successfully tested in an industrial case study.

The first method extended by the authors, the neural-network based grammar inference, RNet algorithm, defines a recurrent neural network architecture which is trained by the sequences of events. After training, this neural net is able to characterize a current system state by looking on past behavior. The authors extract the FSM from the trained neural network by presenting different strings to it and extracting the hidden neurons activity through observations. Due to the nature of Neural Net, closely related activation patterns are clustered into the same state; therefore, by noting the current pattern, the input token, and the next activation pattern, transitions are recorded and compiled into the inferred FSM.

The second method investigated, is a purely algorithmic KTail method, which was taken from the work of Biermann & Feldman [3]. The idea is that a current state is defined by what future behaviors can occur from it. The *future* is defined as the set of next k tokens. By looking at a window of successor events, the KTail algorithm can build the equivalence classes that compose the process model. The authors extensively modified the original KTail algorithm improving the folding in the mined model making it more robust to noise.

The Markov based method developed by the authors is based on both algorithmic and statistical approaches. It takes to account past and future system behavior in order to guess the current system state. Assuming that a finite number of states can define the process, and that the probability of the next state is based only on the current state (Markov property), the authors built a n^{th}-order Markov model using the first and second order probabilities. Once built, the transition probability table corresponding to the Markov model is converted into FSM which is in turn reduced based on the user-specified cut-off threshold for probabilities.

The authors implemented all three of these algorithms in a software tool called DaGama as a plugin for larger software system called Balboa [5]. By performing benchmarking, Cook & Wolf found that the Markov algorithm was superior to the two others. RNet was found to be the worst of the three algorithms. The software tool was applied to a real-world process data and demonstarted an abstraction of the actual process executions and ability to capture important properties of the process behavior. The major backdraw of the approach, as stated by the authors, lies in the inability of the FSMs to model concurrency of processes which limits its applicability to the software development process. Later, Cook et al. in [6] addressed this limitation by using Petri-nets and Moore-type FSM.

Another set of findings relevant to my research approach was developed by Rubin et al. [21] and van der Aalst et al. [22] and is called *incremental workflow mining*. The authors not only designed sophisticated algorithms but built a software system using a business process mining framework called ProM by van Dongen et al. [23] which synthesizes a Petri Net corresponding to the observed process. The system was tested on SCM logs and while the process artifacts retrieved from the SCM system are rather high-level, the approach discussed is very promising for the modeling of software processes from the low-level product and process data.

Within the incremental workflow mining framework, the input data from the SCM log and audit trail information is mapped to the event chain which corresponds to the software process artifacts. The authors call this process *abstraction on the log level* which is implemented as a set of filters which not only aggregates basic events into single high-level entities but also removes data irrelevant to the mining process

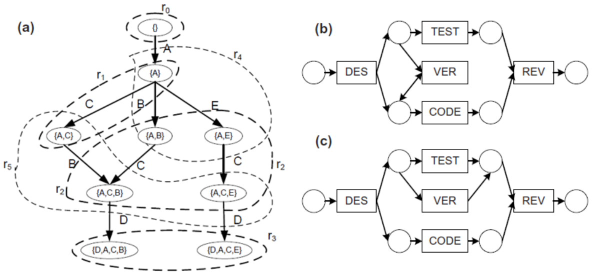


Figure 2: Illustration of the "Generation and Synthesis Approach" from [23]: a) Transition System with regions shown; b),c) Petri Nets synthesized from the Transition System.

(noise).

The event chain constructed through the abstraction is then treated with the *Generate* part of the *"Generate and Synthesis"* [22] algorithm in order to generate a *Transition System* which represents an ordered series of events. This algorithm looks at the history (prefix) and the future (suffix) sequences of events related to the current one in order to discover transitions. When applied to the abstracted log information, the algorithm generates a rather large Transition System graph where edges connect to abstracted events. This transition system is then successively simplified by using various reduction strategies. At the last step of the incremental workflow mining approach, Transition Systems are used to *Synthesize* labeled Petri nets (where different transition can refer to the same event) with the help of *"regions theory"* [8]. As with the Transition System generation, the authors investigate many different strategies of Petri nets synthesis, showing significant variability in the results achieved. (see Figure 2). The significant contribution of this research is in the generality of the method. It was shown that by tuning the "Generate" and "Synthesize" phases it is possible to tailor the algorithm to a wide variety of processes. In particular, as mentioned before, Rubin et al. successfully applied this framework to the SCM logs and audit trails analysis.

In addition to discussed work latest trends in software process research emphasize mining of software process artifacts and behaviors [15] [20].

It worth noting that while reviewed approaches were shown to be applicable for the concurrent processes, to the best knowledge of the author of this article they were not directly applied to the software process artifacts as well as the approach I am taking has never been attempted. This may be partly due to the lack of means of automated, real-time data collection of fine-grained developer behaviors as well as a lack of attention to the produced by Continuous Integration Systems software process artifact trails. In my work, by leveraging the ability of the Hackystat system [16] to collect such a fine grained data, I propose to extend previous research with new knowledge that will support improvements in our understanding of software process.

3. RESEARCH OBJECTIVES

As shown by previous research, it is possible to infer and successively formalize sequential software process by observing its artifacts, and in particular, recurrent behavioral patterns. The problem of finding such patterns is the cornerstone of my research.

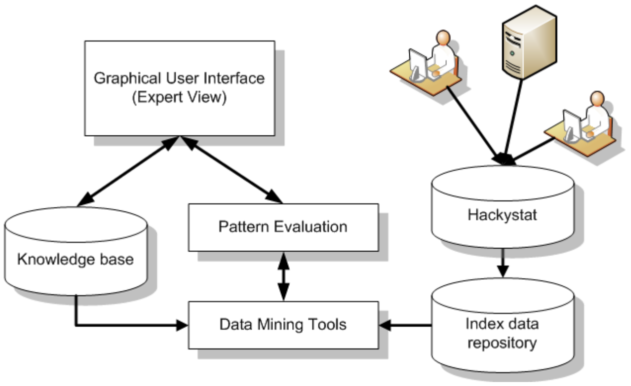


Figure 3: The high-level system overview. Software engineering process and product data are collected and aggregated by Hackystat and then used to generate temporal symbolic indexes. Data mining tools constrained by software engineering domain knowledge are then used for unsupervised patterns discovery. The GUI provides an interface to the discovered patterns and aids in investigation of a discovered phenomena.

The main research objectives of my work is a design, development and evaluation of a previously unexplored approach to discovering of recurrent behaviors in software process through the temporal data mining of low-level process and product artifacts.

In order to achieve the goal I am currently conducting an exploratory study of applicability of pattern-mining techniques to the software process artifacts.

4. RESEARCH APPROACH

My approach to this problem rests on the application of data-mining techniques to symbolic time-point and time-interval series constructed directly from the real-valued telemetry streams provided by Hackystat.

To investigate the requirements for a software tool that aids in the discovery of recurrent behavioral patterns in software process, I am designing and developing the "Software Trajectory" framework. A high-level overview of the framework is shown in Figure 3 and resembles the flow of the "Knowledge Discovery in Database" process discussed by Han et al. in [13]. As shown, the data collected by Hackystat is transformed into a symbolic format and then indexed for further use in data-mining. The tools, designed for data-mining, have a specific restrictions placed on the search space by domain and context knowledge in an attempt to limit the amount of reported patterns to useful ones. I am planning to design a GUI in a way that will allow easy access and modification of these restrictions.

While I intend to use Hackystat as a primary data collection system in my experimental setup, I am currently looking on other ways of data collection and abstraction. The ability to abstract into symbolic representation various software artefact streams such as SCM logs and audit trails is crucial in the exploratory study phase as well as in the evaluation of data-mining techniques efficiency and applicability.

5. DATA AND ANALYSIS METHODS

As mentioned before, this research based on the mining of recurrent behavioral patterns from a symbolic representation of software processes as a temporal sequences of events performed by individual developers or automated tools with or without concurrency.

Various types of data are suitable for software process analyzes. For example, simplest data extracted from SCM logs allows to track software change and perform a basic software process reconstruction - however a very little information can be recalled about development activities and behaviors from such data and models. When development audit trails added to analyzes - enriched data allow to shed more light on the performed process, while adding considerable amount of noise to the models. By using logs of an automated build system it is possible to collect a broader spectrum of artifacts and software metrics such as cyclomatic complexity or coverage, moreover centralized build system creates a set of singular points which are tying together concurrent processes performed by developers. Finally, collecting temporal data about atomic development events by the means of Hackystat and similar systems - such as background compilation and test results, buffer transfers along with quantification of the actual effort enables rich characterization of the dynamic behavior of a software process in great detail. Hovewer, hypotetically, since it was never tested yet, the amount of noise will be unbearable high for proposed by previous work methods such as Finite Automata and Petri Nets (due to the “curse of dimensionality” [14]).

Approach I am taking in my research differs from previous attempts by it’s relative simplicity and inspired by two research fields: first is recent advances in pattern mining in the time-series data and second is advances in motif finding in biological sequence analysis.

Current state of the art approach in temporal data mining called Symbolic Aggregate approXimation and was proposed by Lin et al. in [19]. This method extends the PAA-based approach, inheriting algorithmic simplicity and low computational complexity, while providing satisfiable sensitivity and selectivity in range-query processing. Moreover, as mentioned by the authors, the use of a symbolic representation opens the door to the existing wealth of data-structures and string-manipulation algorithms in computer science such as hashing, regular expression pattern matching, suffix trees etc. As showed by the authors, by using SAX it is possible to index and find various recurrent and “surprise” patterns in vast amounts of data in almost linear time and space [1] [18] [17].

Multiple alignments of protein sequences are important in many biological applications, including phylogenetic tree estimation, secondary structure prediction and critical residue identification. Recent advances in MSA algorithms [10] include a new method for design and evaluation of objective functions for improving alignments by profiling.

While the first approach derived from SAX may fill the bill - if many data will be collected to gather strong evidence exist

6. EMPIRICAL STUDY DESIGN

I propose to conduct two case studies: *Public data case study*, and *Classroom case study* in order to empirically evaluate the capabilities and performance of Software Trajectory framework. These studies differ in the granularity of data used, and in the approaches for evaluation.

My intent behind these empirical studies is to assess the ability of Software Trajectory framework to recognize well known recurrent behavioral patterns and software processes (for example Test Driven Development), as well as its ability to discover new ones. In addition, these studies will support a classification and extension of the current Hackystat sensor family in order to improve Software Trajectory’s performance. It is quite possible that some of the currently collected sensor data will be excluded from the Software Trajectory datasets, while some new ones will be designed and developed in order to capture important features from the studied software development data streams.

The proposed public data case study is based on the use of publicly available Software Configuration Management (SCM) audit trails of the big, ongoing software projects such as Eclipse, GNOME etc. Mining of SCM repositories is a well-developed area of research with much work published [11]. SCM repositories contain coarse software product artifacts which are usually mined with a purpose of discovering of various characteristics of software evolution and software process. I am using a mixed-method approach in this study. In the first phase of this study, I plan to perform SCM audit trail data mining following published work and using Software Trajectory as a tool in order to discover confirmed patterns in software process artifacts, and thus quantitatively evaluate Software Trajectory’s performance when compared to existing tools. In the second phase, I will develop my own pre-processing and taxonomy mapping of software process artifacts into temporal symbolic series. By using this data and Software Trajectory framework, I plan to develop a new approach for SCM audit trail mining and possibly discover new evolutionary behaviors within software process. These discovered knowledge will be evaluated through the peer-reviewed publication submitted for the annual MSR challenge [11].

The classroom case study is based on a more comprehensive data set. This data will be collected by Hackystat from Continuous Integration and from individual developers and will contain fine-grained information about performed software process. The approach I am taking in this study is very similar to the public data case study. I will develop my own taxonomy for mapping of software process artifacts into symbolic temporal data and will apply Software Trajectory analyzes to this data in order to discover recurrent behaviors. In turn, these discovered knowledge will be evaluated through interviewing for usefulness and meaningfulness. Results of interviewing will be used to improve Software Trajectory and will constitute part of my thesis and following publication.

Both case studies are exploratory in nature. At this point of my research, I can only see that the properties of my approach and its current implementation in the Software Trajectory framework appear to be very promising. The wealth of developed techniques for temporal symbolic data mining and recent development of SAX approximation allow me to overcome many computational limitations in existing approaches for mining of software process artifacts. The current implementation of Hackystat provides the ability to capture fine-grain software product and process metrics providing a richness of data, which, potentially, might reveal new insights. Current research in software process discovery indicates the overall feasibility of proposed goals in the discovery of unknown recurrent behaviors in software process.

Nevertheless, there is no prior knowledge about application of these techniques to software process mining. Moreover, at this stage of my research, it is impossible to foresee if new recurrent behaviors will be discovered or their meaningfulness or usefulness for real world applications. For this reason I am undertaking a constructivism paradigm in my research [9], and will develop knowledge about the applicability of my approach to software process mining during the development of Software Trajectory framework and its empirical evaluation. By designing, developing, deploying and observing a software system, and by conducting interviews and surveys, I will gain the desired experience and knowledge.

But it is possible that this part of my research will fail, and I will not be able to discover any meaningful novel knowledge about software process. If so, I will apply every effort to investigate and explain the pitfalls of my approach to the domain of software process data mining. It may be that failure is due to the specificity of domain, or due to the insufficiency of the information enclosed in the collected artifacts, or maybe due to inefficiency of augmented methods. Through thorough analyzes of failed experiments and collection of feedback, I will outline boundaries of the approach taken in this work and appropriate avenues for future development.

7. PRELIMINARY RESULTS

During my work on the pilot version of Software Trajectory framework, I began a set of small experiments in order to aid in the architectural design and algorithms implementation. In addition, these experiments helped me to outline the boundaries of applicability of my approach to certain problems in software engineering. I call these experiments the *Pilot study.*

8. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

9. REFERENCES

[1] *HOT SAX: efficiently finding the most unusual time series subsequence*, 2005.

[2] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. pages 467–483. 1998.

[3] A. W. Biermann and J. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Computers*, pages 592–597, 1972.

[4] R. Conradi. SPI frameworks: TQM, CMM, SPICE, ISO 9001, QIP experiences and trends - norwegian SPIQ project.

[5] J. E. Cook. *Process discovery and validation through event-data analysis.* PhD thesis, Boulder, CO, USA, 1996.

[6] J. E. Cook, Z. Du, C. Liu, A. L. Wolf, and Er. Discovering models of behavior for concurrent workflows, 2004.

[7] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, July 1998.

[8] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving petri nets from finite transition systems. *Computers, IEEE Transactions on*, 47(8):859–882, 1998.

[9] J. W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches (2nd Edition).* Sage Publications, Inc, 2nd edition, July 2002.

[10] R. Edgar. Muscle: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5(1):113+, August 2004.

[11] H. Gall, M. Lanza, and T. Zimmermann. 4th international workshop on mining software repositories (MSR 2007). In *Software Engineering - Companion, 2007. ICSE 2007 Companion. 29th International Conference on*, pages 107–108, 2007.

[12] W. W. Gibbs. Software's chronic crisis. *Scientific American*, September 1994.

[13] J. Han and M. Kamber. *Data Mining, Second Edition, Second Edition : Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems) (The Morgan Kaufmann Series in Data Management Systems).* Morgan Kaufmann, January 2006.

[14] G. Hughes. On the mean accuracy of statistical pattern recognizers. *Information Theory, IEEE Transactions on*, 14(1):55–63, 1968.

[15] C. Jensen and W. Scacchi. Guiding the discovery of open source software processes with a reference model. pages 265–270. 2007.

[16] P. M. Johnson, S. Zhang, and P. Senin. Experiences with hackystat as a service-oriented architecture. Technical Report CSDL-09-07, Department of Information and Computer Sciences, University of Hawaii, Honolulu, Hawaii 96822, Los Angeles, California, February 2009.

[17] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286, 2001.

[18] E. Keogh, S. Lonardi, and B. . Chiu. Finding surprising patterns in a time series database in linear time and space. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 550–556, New York, NY, USA, 2002. ACM.

[19] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, October 2007.

[20] D. Lo, H. Cheng, J. Han, S. C. Khoo, and C. Sun. Classification of software behaviors for failure detection: a discriminative pattern mining approach. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and*

data mining, pages 557–566, New York, NY, USA, 2009. ACM.

[21] V. Rubin, C. Günther, W. van der Aalst, E. Kindler, B. van Dongen, and W. Schäfer. Process mining framework for software processes. pages 169–181. 2007.

[22] 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*, 2009.

[23] B. F. van Dongen, A. K. A. 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. pages 444–454. 2005.