LETTER *Special Section on Empirical Software Engineering*

Quantitative Evaluation of Software Component Behavior Discovery Approach

Cong LIU[†a)], ***Member***

SUMMARY During the execution of software systems, their execution data can be recorded. By fully exploiting these data, software practitioners can discover behavioral models describing the actual execution of the underlying software system. The recorded unstructured software execution data may be too complex, spanning over several days, etc. Applying existing discovery techniques results in spaghetti-like models with no clear structure and no valuable information for comprehension. Starting from the observation that a software system is composed of a set of logical components, Liu *et al.* propose to decompose the software behavior discovery problem into smaller independent ones by discovering a behavioral model per component in [1]. However, the effectiveness of the proposed approach is not fully evaluated and compared with existing approaches. In this paper, we evaluate the quality (in terms of understandability/complexity) of discovered component behavior models in a quantitative manner. Based on evaluation, we show that this approach can reduce the complexity of the discovered model and gives a better understanding.

key words: *quantitative evaluation, component behavioral model, complexity, software process mining*

1. Introduction

Software systems form an integral part of the most complex artifacts built by humans, and we have become totally dependent on these complex software artifacts. Such complex software systems are extremely difficult to maintain. During the execution of a software system, their execution data can be recorded. By fully exploiting the recorded data, one can discover behavioral models describing the actual execution [1] and [2]. The software behavioral model provides insight regarding the real usage of the software, motivates novel idea on model-based testing, enables software usability improvements and redesign, localizes performance problems and architectural challenges [3] and [4].

With the great flush of process mining techniques [5], [6] and [7] on the one hand, and the growing availability of software execution data on the other hand, a new form of software analytic is enabled, i.e., applying process mining techniques to analyze software execution data. This interdisciplinary research area is called *Software Process Mining* [8], [9] and [10], which aims to analyze software execution data from a process-oriented perspective. Applying existing process mining techniques results in flat and spaghetti-like models with no clear structure and no valuable information for comprehension and further anlaysis.

Given the observation that a software system typically involves a set of interacted components. We proposed to decompose the behavior discovery problem into smaller independent ones by discovering a behavioral model per component in [1]. However, the effectiveness of the proposed software component behavior discovery approach is not fully evaluated and compared with existing approaches. In this paper, we propose to evaluate the quality of the software component behavior discovery approach by using a software case study in a quantitative manner.

The remainder of this paper is organized as follows. Section 2 review the basic idea of software component behavior discovery approach. Section 3 shows our quantitative evaluation results. Finally, Sect. 4 concludes the paper.

2. An Overview of the Software Component Behavior Discovery Approach

This section reviews component behavioral model discovery from software execution data as proposed in [1]. The starting point is software execution data, which can be obtained by instrumenting and monitoring software execution. In the following, the main steps are summarized.

- ***Component Execution Data Construction***. Software typically contains a set of *components*. By taking as input software execution data and component configurations, we first construct execution data for each component. Note that the component configurations can be obtained from development documents or identified by clustering classes [11].
- ***Component Instance Identification***. Starting from the software execution data of each component, we propose to identify component instances. The identified instances serve as the basic *case* notion to generate a software event log for each component. Here, a component instance refers to one independent instantiation of a software component.
- ***Hierarchical Software Event Log Construction***. Because a software system usually has a hierarchical structure, the discovered component behavioral model should depict this hierarchy nature. To achieve this, we recursively transform the software event log of each component to a hierarchical one using calling relations among method calls.
- ***Component Behavioral Model Discovery***. For each

Manuscript received February 4, 2020.
Manuscript revised April 23, 2020.
Manuscript publicized May 21, 2020.
[†]The author is with School of Computer Science and Technology, Shandong University of Technology, Zibo 255000, China.
a) E-mail: liucongchina@sdust.edu.cn
DOI: 10.1587/transinf.2020MPL0001

component, we discover a hierarchical Petri net from its hierarchical software event log. Note that we can use the state-of-the-art process discover technique, i.e., *Inductive Miner* [12], in this step.

3. Quantitative Evaluation

In this section, we use an online bookstore software case to show the approach which exploits both component information and hierarchy structure helps to discover better behavioral models in a quantitatively manner. This online bookstore software contains two components: *OnlineBookStore* and *OrderAndDelivery*. The former consists of four classes, i.e., *BookstoreStarter*, *Catalog*, *BookSeller*, *Bookstore*, and the latter involves *Order*class and *Delivery* classes. We first instrument its source code using the open-source *Kieker* framework[†]. therefore, method invocations are stored as software execution data in the XES-software format[††].

Setup. To show the effectiveness of component behavior discovery approach, we conduct four groups of controlled experiments and compare their results with regard to the understandability using a group of quality metrics [13]. Our scope is to show to what extend the use of component information and hierarchy helps to discover more understandable behavioral models. Detailed experiment settings are illustrated in Table 1.

Experiment 1. The first experiment does not use component information and only uses existing *Inductive Miner* [12] to discover a flat Petri net by taking the whole software execution data as input. The discovered behavioral model of this software is shown in Fig. 1. It is a flat Petri net where single-line transitions represent methods and places represent method invocation relation.

Experiment 2. The second experiment does not use component information but uses the hierarchy to discover a hierarchical Petri net by taking the whole software execution data as input. This approach is discussed in [3]. The discovered behavioral model of this software is shown in Fig. 2 where: (1) single-line rectangles represent atomic method calls; and (2) double-line rectangles represent nested method calls which refers to another sub-net. It is worth noting that two kinds of method relations, i.e., *method invocation flow relation* for methods in the same level and *nested method calling relation* for methods of different levels, are contained in the behavioral model. For example, *Bookstore.init()* is followed by *Bookstore.searchBook()* in the second-level model and *Bookstore.getOffers()* is called by *Bookstore.searchBook()*.

The next two experiments use component information to decompose the software behavior model into smaller independent ones by discovering a behavioral model per component. To do so, we first identify component instances for each component and transform the identified software execution data to a software event log. After identification and transformation, we get two software execution data for components *OnlineBookStore* and *OrderAndDelivery*.

[†]http://kieker-monitoring.net/framework/
[††]http://www.xes-standard.org/xesstandardextensions

Table 1 Experiment settings

Experiment	Use Component Information	Hierarchical Behavioral Model
Experiment 1	*No*	*No*
Experiment 2	*No*	*Yes*
Experiment 3	*Yes*	*No*
Experiment 4	*Yes*	*Yes*

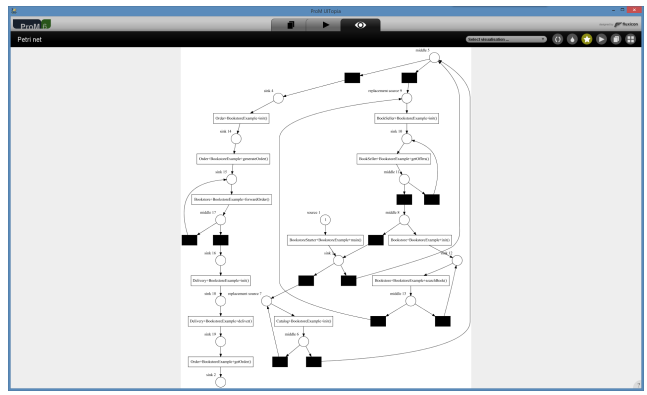


Fig. 1 Software behavior model without hierarchy

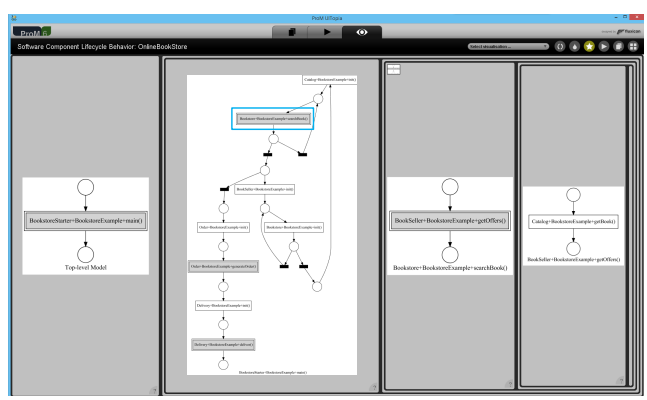
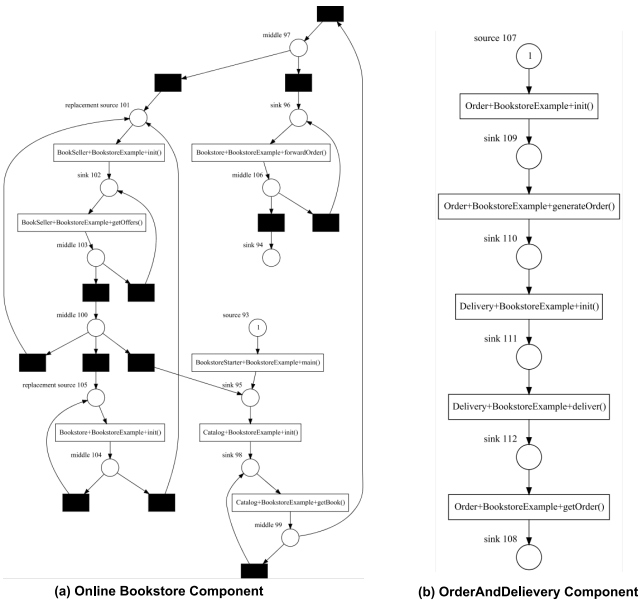


Fig. 2 Software behavior model with hierarchy

Experiment 3. The third experiment uses the component information but only uses existing *Inductive Miner* [12] to discover a flat Petri net for each component. The discovered component behavioral models are shown in Fig. 3.

Experiment 4. The forth experiment uses both component and the hierarchy information as introduced in [1]. The discovered behavior model for component *OnlineBookStore* is shown in Fig. 4. The *OnlineBookStore* component is used to search book stock and get offer for each book. In the *main()* method (left panel of this figure), a loop where each iteration handles the search and get offer separately for each book. The discovered behavior model for *OrderAndDelivery* is shown in Fig. 5. The *OrderAndDelivery* component is used to generate book order and perform delivery. It depicts that this component will generate one book order and perform one delivery for all ordered books in the *Online-*



(a) Online Bookstore Component (b) OrderAndDelievery Component

Fig. 3 Behavior of two components without hierarchy

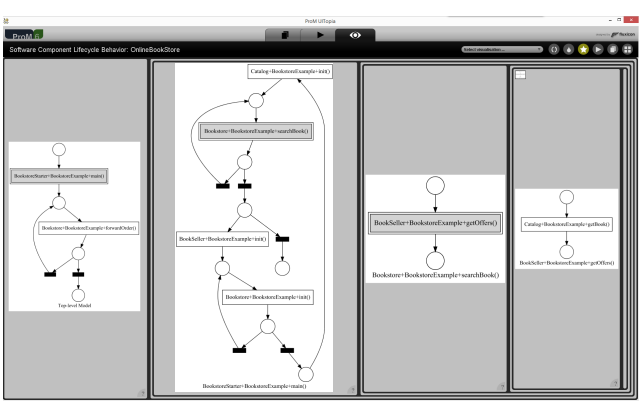


Fig. 4 Behavior model of OnlineBookStore component

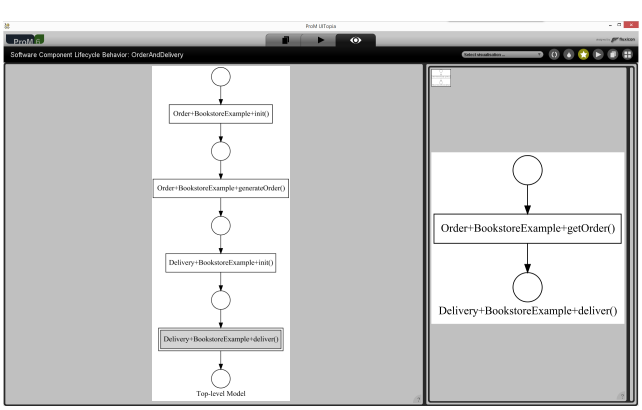


Fig. 5 Behavior model of OrderAndDelivery component

BookStore component.

In the following, we evaluate the quality of the discovered behavioral models in terms of their structural complexity. According to [14], we select the following quality factors as they are regarded as the most convincing ones.

- *Number of Nodes (NoN).*
- *Number of Arcs (NoA).*
- *Control Flow Complexity (CFC).*
- *Average Connector Degree (ACD).*
- *Coefficient of Network Connectivity (CNC).*
- *Density.*

Table 2 Quality results for *Experiments 1-4*

Experiment	*NoN*	*NoA*	*CFC*	*ACD*	*CNC*	*Density*
Experiment 1	*44*	*50*	14	3.17	1.14	0.03
Experiment 2	*40*	*40*	8	3.00	1.00	0.03
Experiment 3	*45*	*50*	13	3.17	1.18	0.04
Experiment 4	*42*	*40*	8	3.00	1.00	0.03

NoN gives the number of nodes in a process model. $NoN = |N|$ where $|N|$ is the number of nodes. *NoA* gives the number of arcs in a process model. $NoA = |A|$ where $|A|$ denotes the number of arcs. The *CFC* metric evaluates the complexity of the process model by XOR-split and AND-split connector and is defined as:

$$CFC = \sum_{c \in AND} 1 + \sum_{c \in XOR} |c^{\bullet}| \tag{1}$$

where *AND* is the AND-split connector set and *XOR* is the XOR-split set. The *ACD* measures the average number of incoming or outcoming arcs an connector is connected to and is defined as:

$$ACD = \frac{1}{|C|} \sum_{c \in C} |c^{\bullet}| + |^{\bullet}c| \tag{2}$$

where C denotes the connector set, $\forall c \in C$, $|c^{\bullet}|$ represents the number of out-coming arcs of c and $|^{\bullet}c|$ represents the number of in-coming arcs of c. The *CNC* gives the ratio of edges to nodes and is defined as:

$$CNC = \frac{|A|}{|N|} \tag{3}$$

where $|A|$ is the number of nodes and $|N|$ is the number of nodes. The *Density* gives the ratio of existing arcs to the maximal number of arcs between the nodes in the model and is defined as:

$$Density = \frac{|A|}{|N| \times (|N| - 1)} \tag{4}$$

The density ranges from 0 to 1. A density close to 1 means that the process graph is highly dense, i.e. all possible connections between the nodes are present.

Table 2 summarizes the quality metrics for the discovered behavioral models of *Experiments 1-4*. Note that *Experiments 3-4* result in two independent models each, and we sum them up to get the overall quality factor values.

It is proved that all these factors point at a negative effect on the model's understandability [14]. According to Table 2, we observe that the model discovered by *Experiment 1*

Table 3 Quality results for *Experiments 2 and 4*

Experiment	*NoN*	*NoA*	*CFC*	*ACD*	*CNC*	*Density*
Experiment 2	*40*	*40*	8	3.00	1.00	0.03
Experiment 4 (onlineBookstore)	*30*	*30*	8	3.00	1.00	0.03
Experiment 4 (orderAndDelivery)	*12*	*10*	0	1.00	0.83	0.07

has roughly the same structural complexity as the model discovered by *Experiment 3*. Similarly, the model discovered by *Experiment 2* has roughly the same structural complexity as the model discovered by *Experiment 4*. However, the complexity values of *Experiment 1* (*Experiment 3*) are bigger than those of *Experiment 2* (*Experiment 4*), which indicates that the use of hierarchy information greatly reduces the complexity of the discovered model and gives a better understanding of how software behaviors.

Different from *Experiment 2*, *Experiment 4* decomposes the whole software behavior into smaller ones. To show the use of component information improve the understandability of discovered models, we compare the quality metrics for the discovered behavioral models of *Experiments 2* and *Experiments 4* (each per component) in Table 3.

According to Table 3, the complexity values of two component behavioral models discovered by *Experiment 4* are smaller than that of the model discovered by *Experiment 2*. In this way, the complexity of the model is reduced greatly, which indicates the use of component information greatly reduces the complexity of the discovered model and gives a better understanding of how software behaviors.

In summary, we demonstrate that the use of component and hierarchy information improve the quality of discovered models from an understandability point of view.

4. Conclusion

By exploiting software execution data, one can discover behavioral models describing the actual behavior of the software system. However, applying existing discovery techniques results in extremely complex models that are difficult for comprehension. A software system is usually composed of a set of components. Starting from this observation, Liu *et al.* propose to decompose the discovery problem into smaller ones by discovering a behavioral model per component in [1]. However, the effectiveness of the proposed approach is not evaluated. In this paper, we evaluate the understandability (or complexity) of discovered component behavior models in a quantitative manner. By experimental evaluation, we show that our approach reduces the complexity of the discovered models and gives a better understanding of how software behaves.

Acknowledgments

This work was supported in part by National Natural Science Foundation of China (61902222), and Taishan Scholars Program of Shandong Province (tsqn201909109).

References

[1] C. Liu, B. van Dongen, N. Assy, and W. van der Aalst, "Component behavior discovery from software execution data," International Conference on Computational Intelligence and Data Mining, pp.1–8, IEEE, 2016.

[2] C. Liu, B. van Dongen, N. Assy, and W. van der Aalst, "Software architectural model discovery from execution data," 13th International Conference on Evaluation of Novel Approaches to Software Engineering, pp.1–8, 2018.

[3] C. Liu, "Automatic discovery of behavioral models from software execution data," IEEE Transactions on Automation Science and Engineering, vol.15, no.4, pp.1897–1908, 2018.

[4] C. Liu, "Software data analytics: Architectural model discovery and design pattern detection," Technische Universiteit Eindhoven, pp.1–299, 2019.

[5] W. Van Der Aalst, "Process mining: discovery, conformance and enhancement of business processes," Springer Science & Business Media, 2011.

[6] C. Liu, Y. Pei, Q. Zeng, and H. Duan, "Logrank: An approach to sample business process event log for efficient discovery," International Conference on Knowledge Science, Engineering and Management, pp.415–425, Springer, 2018.

[7] C. Liu, H. Duan, Z. Qingtian, M. Zhou, F. Lu, and J. Cheng, "Towards comprehensive support for privacy preservation cross-organization business process mining," IEEE Trans. Services Comput., vol.12, no.4, pp.639–653, 2019.

[8] C. Liu, J. Zhang, G. Li, S. Gao, and Q. Zeng, "A two-layered framework for the discovery of software behavior: A case study," IEICE TRANSACTIONS on Information and Systems, vol.101, no.8, pp.2005–2014, 2018.

[9] C. Liu, B. van Dongen, N. Assy, and W. van der Aalst, "Component interface identification and behavioral model discovery from software execution data," International Conference on Program Comprehension, pp.97–107, ACM, 2018.

[10] C. Liu, S. Wang, S. Gao, F. Zhang, and J. Cheng, "User behavior discovery from low-level software execution log," IEEJ Transactions on Electrical and Electronic Engineering, vol.13, no.11, pp.1624–1632, 2018.

[11] C. Liu, B.F. van Dongen, N. Assy, and W.M. van der Aalst, "A general framework to identify software components from execution data," Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering, pp.234–241, SCITEPRESS-Science and Technology Publications, Lda, 2019.

[12] S.J.J. Leemans, D. Fahland, and W. van der Aalst, "Discovering block-structured process models from event logs-a constructive approach," Application and Theory of Petri Nets and Concurrency, vol.7927, pp.311–329, Springer, 2013.

[13] J. Mendling, Detection and prediction of errors in EPC business process models, Ph.D. thesis, Wirtschaftsuniversität Wien Vienna, 2007.

[14] J. Mendling, H.A. Reijers, and J. Cardoso, "What makes process models understandable?," Business Process Management, pp.48–63, Springer, 2007.