Fuzzy-based process mining to discover the coding behavior: challenges and future works*

Pasquale Ardimento[1,*,†], Lerina Aversano[2,†], Mario Luca Bernardi[2,†] and Marta Cimitile[3,†]

[1]University of Bari Aldo Moro, 4 Orabona Street, Bari, Italy
[2]Unisannio University, Department of Engineering, Palazzo ex Poste, Benevento, Italy
[3]UnitelmaSapienza University, Rome

Abstract

Discovering the coding behavior of programmers is an emerging application domain in the process mining field. Comprehension of how programmers head the coding of software have a strong potential to better support the coding workflow. In our previous work, we introduced and evaluated an environment, called CodingMiner, to generate event logs from IDE usage enabling the adoption of fuzzy-based process mining techniques to model the programmers' coding process. The mined processes have shown different IDE usage patterns for programmers with different skills and performances. Our approach, currently, is not able to represent the behavior concerning the usage of programming core constructs such as, in the case of Object-Oriented paradigm, abstraction, object state and behavior. In this paper, we are interested to discuss the main research challenges and sketch possible actions to adopt for improving the realization of the proposed environment to also represent the behavior in using the core constructs.

Keywords

Process Mining, Fuzzy Miner, Coding Behavior, Programmer Activities,

1. Introduction

The comprehension of software coding processes is not a simple task. By nature, these processes are profoundly iterative and characterized by a very loose ordering of their activities [1]. Programmers, starting from a model of the problem, write the source code by applying best practices of programming, and core structures and principles of the programming language paradigm [2]. This involves the hierarchical breakdown of the source code into smaller components, and, in turn, the choice or application of core structures and principles to implement such components [3]. This is a complex process mainly based on human creativity where

OLUD 2022: First Workshop on Online Learning from Uncertain Data Streams, July 18, 2022, Padua, Italy.
*Corresponding author: Pasquale Ardimento
†These authors contributed equally.
pasquale.ardimento@uniba.it (P. Ardimento); aversano@unisannio.it (L. Aversano); bernardi@unisannio.it (M. L. Bernardi); marta.cimitile@unitelmasapienza.it (M. Cimitile)
https://www.uniba.it/it/docenti/ardimento-pasquale/ (P. Ardimento); https://www.unisannio.it/user/596/ (L. Aversano); https://www.unisannio.it/user/12387 (M. L. Bernardi); https://www.unitelmasapienza.it/it/contenuti/personale/marta-cimitile (M. Cimitile)
0000-0001-6134-2993 (P. Ardimento); 0000-0003-2436-6835 (L. Aversano); 0000-0002-3223-7032 (M. L. Bernardi); 0000-0003-2403-8313 (M. Cimitile)

the attitude, knowledge, and experience of programmers have a great impact on the resulting behavior during coding. Moreover, differences on coding behavior are not strictly related to the level of attitude, knowledge, and experience of programmers.

> Even among programmers of very similar experience levels, differences of as much as 100 to 1 were found across programmers in the time taken to write a given program. Additionally, across problems constructed to be of similar difficulty, an individual programmer often displayed a six-fold difference in writing time[4].

The awareness of the above-discussed critical issues suggests the study of process mining techniques to understand the behavior of the software coding, as many studies on the subject testify [5, 6, 7, 8, 9, 10]. Furthermore, when the programmers are students, understanding the coding behavior is valuable to predict their outcomes [11]. In a previous work [12], we mainly focused on using a fuzzy-model process mining approach to understand the behavior of the programmers in writing the software source code. In this regard, in [12] we defined an environment, called CodingMiner, to generate event logs from IDE usage enabling the adoption of fuzzy-based process mining techniques to study the programmers' coding process. We executed an empirical evaluation using logs from the coding sessions of students attending the 2nd-year of a BSc degree in computer science. By using the CodingMiner, we highlighted emergent and interesting programmers' behaviors during coding. The mined processes show different IDE usage patterns for programmers with different skills and performances. This environment, however, is not able to represent the different usage of language constructs, such as the ones above mentioned. In this paper, we are interested to discuss the main research challenges and sketch possible actions to adopt to improve the realization of the proposed environment and, in general, of a fuzz-based process mining approach to discover the coding behavior.

Specifically, this paper briefly describes how the CodingMiner environment is defined, then discusses research challenges to improve the discovery of the coding behavior and, finally, points out possible approaches to address these challenges.

The rest of the paper is structured as follows. In Section 2 the CodingMiner environment is introduced whereas in Section 3 challenges and guidelines to improve the realization of this framework. Section 4 provides paper conclusions.

2. The CodingMiner Environment

Figure 1 gives an overview of the workflow mining approach based on IDE instrumentation. The core component is the Log Processor, a plugin for the Eclipse IDE that extends logging and monitoring capabilities of the development environment. As shown in the figure 1, while programmers interact with the development environment to implement a software system, they generate a stream of human-computer interaction (HCI) events. The Log Processor is triggered to capture these interactions. CodingMiner collects all these interactions and stores them as event logs in a central repository. The collected logs, when stored, need to be refined to apply process mining. Specifically, the following steps are executed:

1. interactions that are unrelated with projects in the programmer workspace are removed;

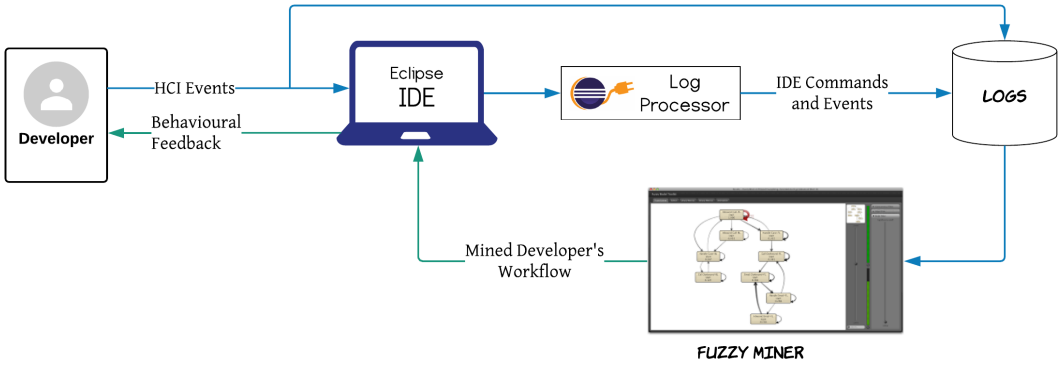

Figure 1: The CodingMiner environment architecture.

2. inconsistencies in the data are detected and corrected;
3. the low-level logs are converted to the XES[1] event log model;
4. low-level and high-level event streams are integrated.

The refined and combined events stream can be analyzed by the Fuzzy Miner [13] to mine the coding process executed by programmers.

These mined processes are shown, within the CodingMiner environment, to provide feedback to programmers about their behaviors during coding. In the following sub-sections, the Log Processor (Section 2.1) and the Fuzzy Miner (Section 2.2) components are briefly discussed. More details about CodingMiner environment are in [12].

2.1. The Log Processor

The Log Processor is an extension of the Fluorite plugin, an open-source instrumentation plugin, capable of recording low-level programmers' interactions with the IDE without interrupting the coding activities. Log Process tracks both low-level events (e.g. mainly keyboard key presses, shortcuts, mouse movements and gestures) adding contextual information and high-level events (e.g., all the commands issued at IDE level, like create or open a file, close a project, open a view, reset perspective, etc.).

For each single coding session performed by a single programmer a synthetic id to group events is recorded. Moreover, all the collected events have related to involved resources (including file resources). In this way, coding events are linked to the programmer's actions on resources in the IDE.

The Log Processor is also able to capture IDE commands (i.e., an action or command issued by the programmer to the IDE). It is also able to model a specific kind of interaction among the programmers and contains contextual information (e.g., the involved resources and their possible state changes). The events captured can be categorized as follows:

- **IDE Commands**: all global commands issued to IDE (e.g. open a view, switch a perspective or accessing a resource like a file);

[1]http://www.xes-standard.org

- **Editor Commands**: all activities happening in the editor (e.g., cut and paste commands, text selection and modification and all actions regarding code writing);
- **Debugging Events**: all debugging activities (e.g., breakpoints definition, whatches on variables and their inspection at runtime, debug profiles definitions, etc.);
- **Refactoring Events**: all the commands related to refactoring activities (e.g., selecting a refactoring among the one available, istructing the IDE on how to perform it and launch its execution).

2.2. Fuzzy Miner Component

The collected events logs are finally mined by the Fuzzy Miner included in the ProM toolkit [13].

The Fuzzy Miner takes as input the logs of the programmer activities captured by the Log Processor and creates an appropriate representation of the development processes expressed in the mined log. Fuzzy Miner is particularly suitable for mining less-structured processes exhibiting unstructured and flexible behavior, like development process tend to be. The Fuzzy Miner is based on the main idea that some kinds of processes are better represented using adaptive techniques providing explicit flexibility [13]. It represents the mined process using a fuzzy model that is *deliberately* imprecise to omit behavior that has low significance or is not correlated with interesting patterns.

3. Challenges and Guidelines

Each step of the CodingMiner environment presented in Fig. 1 gives rise to research challenges. In the following, we give an overview of some of these challenges and propose approaches to tackle them.

- **Recording**. The main challenge in this step is to identify what actions must be recorded. The same action (e.g., InsertString command) can either be important or irrelevant in a given context. For example, typing text for adding a new method is an important event while typing text for an inline comment is an irrelevant event. For this reason, when a programmer makes a change it is necessary to know both the element type (e.g. class, interface, field, subclass, etc) involved and the change type made. Examples of change type for a class are: change of accessibility, add/remove/change inheritance, add/remove/change attribute, add/remove/change comment, add/remove/change attribute, etc. Furthermore, capturing information about element and change type could help to construct a process model able to represent if and when the object-oriented language core topics have been used. Object-Oriented core topics are design, abstraction, hierarchy, typing, and encapsulation [14].
  For example, Object-oriented design is meant as decomposition into objects carrying state and having behavior. A process model representing the coding behavior should represent when and how many properties of an object have been defined as well as when and how many methods have been defined. Furthermore, without information on element and

change type, the process model will only reflect the way the programmers use the IDE to write source code but not the way they code.
Existing coding event recording plugins, also including CodingMiner, are not actually able to capture this information. For example, the Eclipse plugin developed by Caldeira et al. [5], capable of listening to the actions programmers executes, aims to support the discovery of the coding processes and compare them in terms of efficiency and effectiveness. The authors evaluated the proposed approach on subjects attending the 3rd year of a BSc degree on computer science. The results obtained give some evidence that teams' proficiency can be inferred by analyzing mined process models representing their behavior. However, the plugin captures only events within a project context and generic events at the Eclipse global context. This implies that the behavior observed concerns the usage of Eclipse IDE and not properly the developers' coding behavior. In [8], the authors declare to detect both element and change type. The authors developed a constructor that classifies a source code history by fine grained changes and constructs an event log file. They used the Process Mining approach, the Inductive Miner process discovery algorithm, to understand the way programmers perform code production activities. A preliminary evaluation has been performed involving only three students in developing a program made up of one class. Unfortunately, at this time it is not possible consider the process model constructed as significant because it only represents the coding workflow of a single method defined in a class. In a real world coding process, instead, the coding workflow has to represent a more complicated reality made up of classes, sub-classes, abstraction, accessibility and so on. In [15] the authors developed a library, called Entry, for generating log of programmer's behavior in the process of block programming and defined required common items in creating block log process. This library [16] is able to capture several information such as, for example, "the number of times blocks / scenes / objects are created and used", the number of times modifying conditional expression/internal variable of block", "operation time taken to finish the goal resolve the problem", etc. This library could be used to construct tables of frequencies, durations and other statistics and also process models, by applying process discovery techniques but, also in this case, the models constructed would be not able to distinguish relevant from irrelevant situations.

- **Noise filtering**. One of the main challenges of this stage is to separate noise from events that contribute to tasks. In a coding session noises can be represented by activities that can occur spontaneously at any point in the execution. Such activities, called chaotic activities [17], impacts the quality of the resulting process models obtained with process discovery techniques. For example, a searching activity can occur at any point for any task in the execution. To filter out such chaotic events, in [17] four novel promising techniques, rooted in information theory and Bayesian statistics, are described. The authors have shown, through experiments on seventeen real-life datasets, that all four proposed activity filtering techniques outperform frequency-based filtering on real data and that in all cases the performance is highly dependent on the characteristics of the event log. This means, however, that the ultimate decision on which activities to include has always to be supervised by the final user.
- **Segmentation**. A coding session log, in its raw form, consists of one single sequence of

events recorded during a session. During this session, a programmer may have performed several executions of one or multiple tasks. In other words, a coding session log may contain information about several tasks, whose actions and events are mixed in some order that reflects the particular order of their execution by the programmer. Moreover, the same task can be ”spread” across multiple logs, for example the creation of a class can be performed on several logs. A possible solution could consist in segmenting a coding session log into traces, such that each trace to one execution of a task (e.g. the definition of a class, the definition of an interface, etc).

- **Simplification**. Coding process is not executed within rigid, inflexible workflow management systems and the like, which enforce correct, predictive behavior. Programmers write their source code mainly based on their knowledge, skills, and experience. Such a process does not enforce any defined behavior at all, in a somewhat it describes a more ”loose” manner that does not strictly define a specific path of execution. It is obvious that executing such a process within such less restrictive environments will lead to more diverse and less-structured behavior. This abundance of less-structured observed behavior leads to construct “spaghetti” process models. In this cases a possible solution is represented by using the fuzzy miner approach. The Fuzzy Miner controls such imprecision by means of two metrics: significance and correlation. In particular, significance can be determined both for events and precedence relations over them: it provides us with a measure of the relative importance of behavior. It specifies the level of interest we have in events occurring in well-defined control flow conditions (e.g. precedence relationships, chain of events, and other relationships). For instance, frequency measurement or precedence constraints are a way to measure significance. Correlation is important when studying precedence constraints over the event stream. It measures how two events, following one another, are closely related. Based on these two metrics, which have been defined specially for this purpose, we can sketch the approach, proposed in the CodingMiner environment, for process simplification as follows:
  - more significant behaviors are preserved;
  - low significant but highly correlated behaviors are aggregated;
  - Less significant and lowly correlated behaviors are abstracted.

  Anyway, the Fuzzy Miner is not sufficient. Even if an event belongs to a task, it may still be redundant. For example, when a programmer defines the name of a new method with a mistake, and then he immediately renames it. In this case, the events that belong to the second time of naming the method are redundant. Depending on the context, the same event may be integral part of a routine or it may be redundant. Thus, classical frequency-based filtering approaches, like [18], cannot be applied to address this problem. One of the possible solutions is to use sequential pattern mining techniques to distinguish between events that are part of mainstream behavior and outlier events [19]. However, in case some events are rarely seen during a task execution they can be mistakenly treated as outliers. The outlined problem creates a need for semantic filtering. Groups of events can be combined into actions of a higher semantic meaning. The challenge here is to identify the semantic boundaries of an action and the attributes to form its payload.

4. Conclusions

In previous work, we have exposed an environment, called CodingMiner, capable of analyzing coding logs of fine-grained programmer interactions with Eclipse IDE system, for Java application, to represent the coding behavior. We have already applied the CodingMiner in a CS2 course obtaining encouraging but not completely satisfactory results. As a preliminary step to improve this environment, here we sketched challenges that need to be overcome to improve the CodingMiner's components and, in general, the fuzzy-based process mining approach to discover the coding behavior. We also provided some guidelines to tackle these challenges. One of the key challenges consists in how to discover the object-oriented language core topics in coding activities. Each action has to be associated to the element involved and the change type applied. We belief that, with this capability, our approach will become more meaningful, and applicable for teachers and programmers.

References

[1] R. Guindon, B. Curtis, Control of cognitive processes during software design: What tools are needed?, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '88, ACM, New York, NY, USA, 1988, pp. 263–268. URL: http://doi.acm.org/10.1145/57167.57211. doi:10.1145/57167.57211.

[2] R. Brooks, Towards a theory of the cognitive processes in computer programming, Int. J. Hum.-Comput. Stud. 51 (1999) 197–211. URL: http://dx.doi.org/10.1006/ijhc.1977.0306. doi:10.1006/ijhc.1977.0306.

[3] N. Pennington, A. Y. Lee, B. Rehder, Cognitive activities and levels of abstraction in procedural and object-oriented design, Human-Computer Interaction 10 (1995) 171–226. doi:10.1080/07370024.1995.9667217.

[4] R. Brooks, Towards a theory of the cognitive processes in computer programming, International Journal of Man-Machine Studies 9 (1977) 737–751. URL: https://www.sciencedirect.com/science/article/pii/S0020737377800394. doi:https://doi.org/10.1016/S0020-7373(77)80039-4.

[5] J. Caldeira, F. Brito e Abreu, J. Reis, J. Cardoso, Assessing software development teams' efficiency using process mining, in: 2019 International Conference on Process Mining (ICPM), 2019, pp. 65–72. doi:10.1109/ICPM.2019.00020.

[6] J. Caldeira, F. B. e Abreu, J. Cardoso, R. Ribeiro, C. M. L. Werner, Profiling software developers with process mining and n-gram language models, CoRR abs/2101.06733 (2021). URL: https://arxiv.org/abs/2101.06733. arXiv:2101.06733.

[7] V. Shynkarenko, O. Zhevago, Visualization of program development process, in: 14th IEEE International Conference on Computer Sciences and Information Technologies, CSIT 2019, Lviv, Ukraine, September 17-20, 2019, Volume 2, IEEE, 2019, pp. 142–145. URL: https://doi.org/10.1109/STC-CSIT.2019.8929774. doi:10.1109/STC-CSIT.2019.8929774.

[8] V. Shynkarenko, O. Zhevaho, Application of constructive modeling and process mining approaches to the study of source code development in software engineering courses,

Journal of Communications Software and Systems 17 (2021) 342–349. URL: https://doi.org/10.24138/jcomss-2021-0046. doi:10.24138/jcomss-2021-0046.

[9] M. Leemans, W. M. van der Aalst, Process mining in software systems: Discovering real-life business transactions and process models from distributed systems, in: 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), volume 00, 2015, pp. 44–53. URL: doi.ieeecomputersociety.org/10.1109/MODELS.2015.7338234. doi:10.1109/MODELS.2015.7338234.

[10] C. Liu, B. F. van Dongen, N. Assy, W. M. P. van der Aalst, Component behavior discovery from software execution data, in: 2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016, 2016, pp. 1–8. doi:10.1109/SSCI.2016.7849947.

[11] G. Casalino, G. Castellano, G. Zaza, Neuro-fuzzy systems for learning analytics, in: A. Abraham, N. Gandhi, T. Hanne, T.-P. Hong, T. Nogueira Rios, W. Ding (Eds.), Intelligent Systems Design and Applications, Springer International Publishing, Cham, 2022, pp. 1341–1350.

[12] P. Ardimento, M. L. Bernardi, M. Cimitile, G. D. Ruvo, Learning analytics to improve coding abilities: a fuzzy-based process mining approach, in: 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2019, pp. 1–7. doi:10.1109/FUZZ-IEEE.2019.8859009.

[13] C. W. Günther, W. M. P. van der Aalst, Fuzzy mining – adaptive process simplification based on multi-perspective metrics, in: G. Alonso, P. Dadam, M. Rosemann (Eds.), Business Process Management, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 328–343.

[14] A. f. C. M. A. Joint Task Force on Computing Curricula, I. C. Society, Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, Association for Computing Machinery, New York, NY, USA, 2013.

[15] R.-J. Moon, K.-M. Shim, H.-Y. Lee, H.-J. Kim, Log generation for coding behavior analysis: For focusing on how kids are coding not what they are coding, in: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2017, pp. 575–576.

[16] entryjs library, https://github.com/entrylabs/entryjs/, 2016. [Online; accessed 14-June-2022].

[17] N. Tax, N. Sidorova, W. M. van der Aalst, Discovering more precise process models from event logs by filtering out chaotic activities, Journal of Intelligent Information Systems 52 (2019) 107–139.

[18] R. Conforti, M. L. Rosa, A. H. t. Hofstede, Filtering out infrequent behavior from business process event logs, IEEE Transactions on Knowledge and Data Engineering 29 (2017) 300–314. doi:10.1109/TKDE.2016.2614680.

[19] M. F. Sani, S. J. v. Zelst, W. M. van der Aalst, Improving process discovery results by filtering outliers using conditional behavioural probabilities, in: International Conference on Business Process Management, Springer, 2017, pp. 216–229.