Towards Machine Learning for Learnability of MDD tools *

1st Saad Bin Abid
Model-based Systems Engineering (MbSE)
fortiss GmbH.
Munich, Germany
abid@fortiss.org

2nd Vishal Mahajan
Technical University of Munich
Munich, Germany
vishal.mahajan@tum.de

3rd Levi Lúcio
Model-based Systems Engineering (MbSE)
fortiss GmbH.
Munich, Germany
lucio@fortiss.org

***Abstract*—Learning how to build software systems using new tools can be a daunting task to anyone new to the job. This is especially true of tools that provide a large number of functionalities and views on the system under development, such as IDEs for Model-Driven Development (MDD). Applying Machine Learning (ML) techniques can help in this state of affairs by pointing out to appropriate next actions to rookie or even intermediate developers. AutoFOCUS3 (AF3) is a mature MDD tool we are building in-house and for which we provide regular tutorials to new users. These users come from both the academia (e.g, students/professors) and the industry (e.g. managers/software engineers). Nonetheless, AF3 remains a complex tool and we have found there is a need to speedup the learning curve of the tool for students that attend our tutorials – or alternatively and more importantly for others that simply download the tool and attempt using it without human supervision. In this paper, we describe a machine learning-based recommendation system named MAGNET for aiding beginner and intermediate users of AF3 in learning the tool. We describe how we have gathered data and trained an ML model to suggest new commands, how a recommender system was integrated in the AF3, experiments we have run thus far, and the future directions of our work.**

AF3– MAGNET demo video: https://tinyurl.com/y5skeeks

***Index Terms*—Model-Driven Development (MDD), AutoFOCUS3, Machine Learning, Intelligent Recommendation Systems (IRS), Eclipse IDE, Domain-Specific Languages () development interaction data**

I. INTRODUCTION

Modern IDEs are extremely rich in functionality. Environments such as the Eclipse IDE [4] or the whole range of tools proposed by JETBRAINS [6] are complex, offer an ever-increasing amount of functionalities and are highly customizable. Clearly, over the past decade, IDEs have become increasingly competent at their jobs and help the user in a smart manner, by offering intelligent auto-completion, contextual quick fixes and intentions that help with development productivity while lowering learning curves.

For the past 15 years, we have been developing at fortiss the Eclipse-based AF3 tool [14], [28], for building embedded systems. AF3 is mature and stable and includes articulated perspectives for the complete lifecycle of the development of embedded systems: from requirements, to architecture, deployment, code generation, simulation and verification.

The goal of AF3 is to demonstrate the feasibility, applicability and relevance of MDD tools. It is an open source tool with a 6-month release period and has served and serves as a means to demonstrate state-of-the-art MDD technology. It is also used as a boiler plate to develop proof-of-concept projects with industrial partners [15], [16], [22], [23].

We have recently started offering a tutorial on AF3 [2] to the academic community and to partner or interested companies. The tutorial involves creating and deploying control software onto a real vehicle and implies manipulating different types of software development perspectives (mostly graphical but also textual) via the Eclipse IDE. Given the learning the necessary IDE manipulations requires effort and constant attention and explanations from dedicated human tutors, we have decided to develop an intelligent and non-invasive automated tutor that can be invoked at the press of a button.

Because AF3 offers interconnected visual (model-based) perspectives of the logic of a system under development, providing automated help in such settings differs from doing so for code-based IDEs. Similar tools to AF3 are MATLAB SIMULINK [7] or LABVIEW [10]. AF3 also partly falls in the category of Domain-Specific Language (DSL) workbenches such as MPS [6], METAEDIT+ [9] or ATOM3 [21] – although building DSLs in AF3 must be done programmatically and outside the tool itself, by using EMF facilities.

In this paper we report on our first steps in the usage of ML to aid beginner students in the discovery and understanding of the AF3 tool. In particular, we have implemented an Intelligent Recommendation System (IRS) called MAGNET, that suggests next steps in the context of an AF3 tutorial. The next steps are shown to the user as short videos illustrating the completion of a low-level task, e.g. building states, transitions or simulating the system.

This article is organised as follows: in section II we present the main functionality of the tool, as seen from the point of view of an AF3 developer. Section III details how we have trained and deployed a recommender system in AF3. We then go on to discuss some preliminary results in section V and to place our work regarding the state-of-the-art in section IV. We conclude with future directions in section VI.

DOI 10.18293/SEKE2019-050

II. HIGHLIGHTS

The MAGNET tool is at its core an Intelligent Recommendation System. Core to MAGNET are:

- a *multiclass classification* model trained by using a machine learning algorithm using anonymized sessions of previous students of AF3, and
- recommendation videos that illustrate accomplishing certain tasks in AF3.

The Multiclass classification model in the IRS predicts, with a very short delay, the state of advancement of the student in the tutorial. The prediction is based on the previous interactions of the student with AF3, as well as well as on the previously learned *multiclass classification* model. Based on the predicted state, the user is then presented with relevant hints that suggest ways of continuing the tutorial – in the form recommendation videos that demonstrate common low-level of AF3. The videos themselves independent from the tutorial and were built by the Human-Centered Engineering department at fortiss [5] having in mind further reuse for an enlarged IRS. The recommendation videos illustrate common modelling tasks such as how to create a component or a state automaton, how to add transitions or action code to the transitions of an automaton or how to simulate a given software model. The aim is that the new users quickly familiarize themselves with the common visual manipulations of the tool while going through the tutorial.

The recommendation videos can be accessed by a user of AF3 in two ways:

- Via a *"Help Me"* button on the AF3 toolbar (as illustrated in figure 1). In this case the IRS proposes three recommendation videos that were top-ranked by the multiclass classification algorithm. After the press of the button, the top ranked video starts playing. If desired, the user can switch to the other two videos in the order in which they were ranked by the IRS. The time taken by the IRS to display the video after the press of the *"Help Me"* is under 2 seconds.
- Through a global help button. In this case the user does not neccessarily want the IRS to predict next tasks, but only to provide a full list of possible help videos with common functionalities of AF3. This global help can be accessed via the menu item "Show All Hints" in the AF3 3 tool bar, under the "AI assistance" drop down menu.

We have also implemented the means to activate/deactivate data acquisition directly through the AF3 interface, keeping in mind data privacy and that some students may not want to have their interaction data stored.

III. MACHINE LEARNING PIPELINE

The user-assistance functionality we wish to implement in MAGNET is providing tips to the user based on a predicted next step in the tutorial. In order to achieve this, it is required to predict the next step of the user and then to provide an adequate hint for it, from the set of predefined short videos. The machine learning pipeline we have implemented to achieve this consist of the following steps: 1) Data collection, 2) & 3) Data processing including data cleaning, feature selection and data labelling, 4) ML model selection consisting of model training, validation and testing and 5) model deployment. This pipeline is depicted in figure 2 and each of its steps is discussed in the subsequent paragraphs.

*1) **Data Collection**:* In order to acquire the user interaction data, we instrumented the AF3 framework such that interactions of a user with the tool's graphical user interface are recorded. The instrumentation is achieved using SWTBOT [13] and does not interfere with AF3's functionality. Specifically, we have added an SWTBOT plugin on top of AF3. The plugin starts along with the AF3 tool as a server that listens to events that occur in widgets of AF3's graphical user interface (GUI). Additionally, we have implemented a thin client that communicates via sockets with the SWTBOT server and records to a file all such interactions. The raw data was collected from 11 users during one tutorial session. Altogether, the collected files contain approximately 28000 lines where each line denotes a specific action at a time instance.

*2) **Data Cleaning**:* Each line in the collected data contains UNIX Epoch time in milliseconds followed by a string for each of the user's actions/ interaction. The string consists of two sub-parts, a) a GUI contextual description of the user interaction, and b) a general description of the user action initiated via mouse/ keyboard inputs. During analysis of the data, several characteristics of the data were discovered. Some of the actions recorded in the file are correlated with the preceding actions: for instance, closing of a window is mostly followed by default relocation to last visited window. The same trend is observed with respect to few other attributes in the data. Therefore, the data was filtered to remove such correlations and reduce dimensionality, while ensuring minimum loss of significant information. The filtered data consists of approximately 5000 lines.

*3) **Feature selection and Labelling**:* After having achieved a basic understanding of the data, two features were extracted from the data namely, **Action** (mouse action/ key board action) and **Property** (task dependent description of users state) in a time ordered sequence. A single line of data in isolation at a particular time may be of a practical use from the point of view of task prediction. This is because each line defines a user action at lower/ micro level e.g., one line may indicate opening of a GUI window. On the other hand, a task is defined at a higher level of abstraction which contains lower level actions and conveys a practical meaning such as for example: opening of a window, followed by defining parameters therein and finally moving to a different window perspective. It therefore became essential to label the data in relation to a set of specific AF3 tasks – the classes to be used by the ML classification algorithm. This forms the basis for the machine learning model to understand the lower level data, recognize the current task and predict the next task at an adequate level of abstraction. Nine labels relevant to the tutorial were defined. The labels consist of user tasks such as *navigating to component architecture*, *creation and specification (definition)*

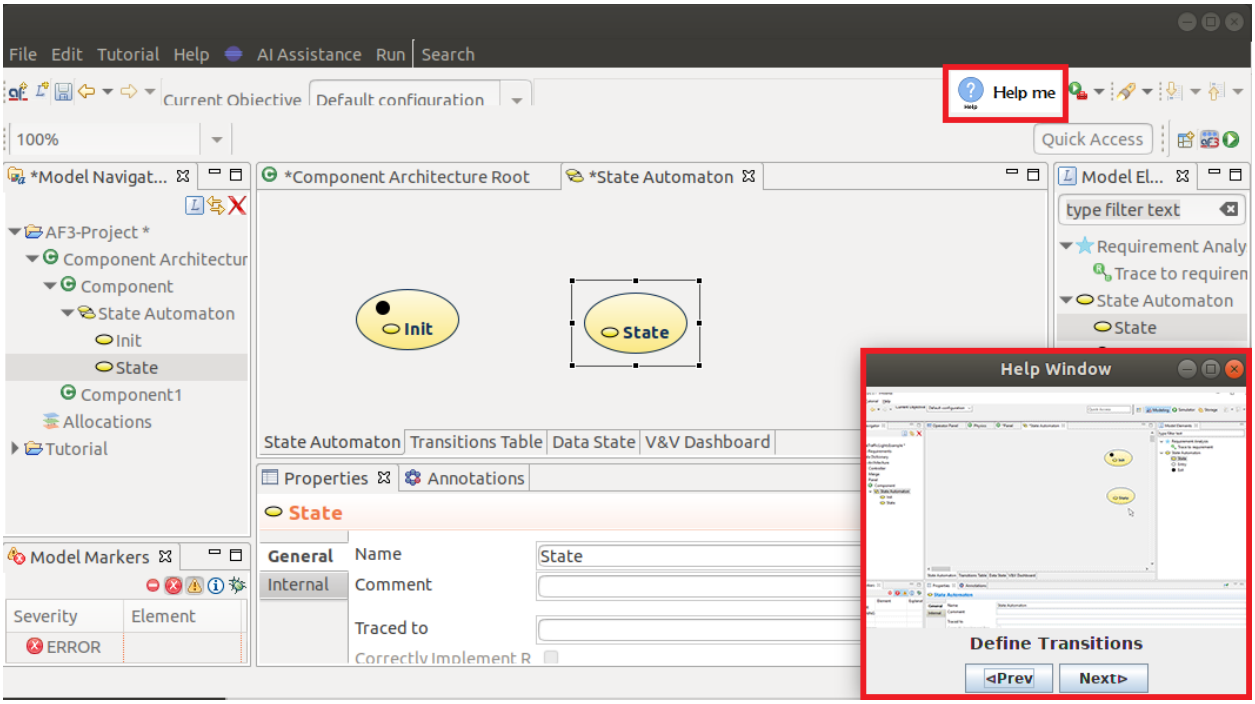

Fig. 1. Recommendation Video in AF3 using "Help Me"

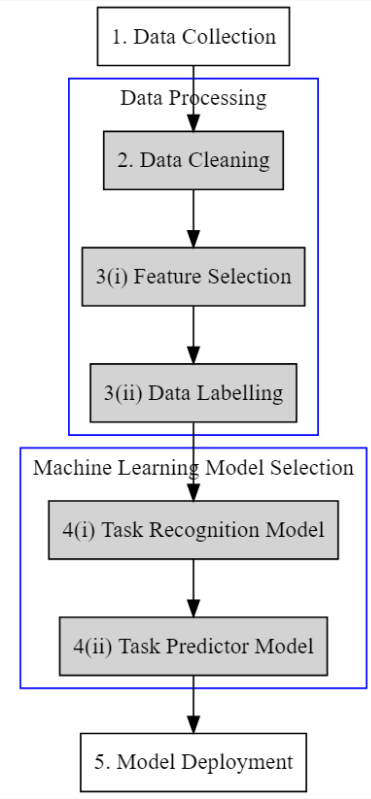

Fig. 2. Machine Learning Pipeline

of components, *specification of transitions*, *creation of ports*, *writing code* and *simulation*. While defining labels, we kept in mind that these labels should neither be too generic nor too specific, in order to aim for a reasonable granularity that a machine learning algorithm can meaningfully build a model for. The data was labelled manually (approx. 2800 lines) after correlating features in the data with specific tasks. In some cases, when the tasks could not be specifically identified from the features, preceding and succeeding labels were used to infer the current task.

*4) **Model Selection**:* Learning an objective during the tutorial can be understood as a multi-classification problem based on the features described above. Two levels of abstraction are necessary for predicting the next state of user. The visual representation of the two models is shown in figure 3.

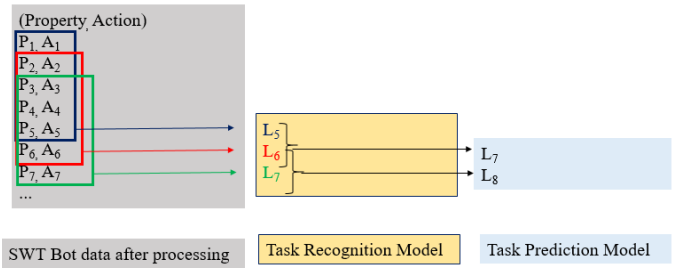

Fig. 3. Visual repsentation of the models (Recognition and Prediction)

The first level model referred to herein as **Task Recognition Model** takes the input consisting of features: Property (P)

and Action (A) in a time sequence of length u and predicts the corresponding label (L). Figure 3 shows how the **Task Recognition Model** uses a sequence of 5 previous steps i.e., $u = 5$ to predict the current task (label) at a given time t. These predictions act as an input for the second level model. The second model referred herein as **Task Predictor Model** uses the sequence of output labels of length v from Task Recognition Model as an input and predicts the next task of the user at future time step $t+1$ as an output. As an example, figure 3 shows **Task Predictor Model** uses a label sequence of 2 previous steps i.e., $v = 2$ to predict the label L at future timestep $t+1$.

Three different machine learning models were tried for training the task recognition model. Due to the similarity of the problem with sequence to sequence prediction, an LSTM model was the first choice [24], [31]. The labelled data was split into training and validation sets in a proportion of 80:20. The LSTM model achieved an accuracy of 20% on the validation data in 100 iterations whereas the training accuracy was very high. The results of LSTM model are shown in figure 4. We attribute the poor performance of the LSTM model on validation data to the model overfitting the training data.

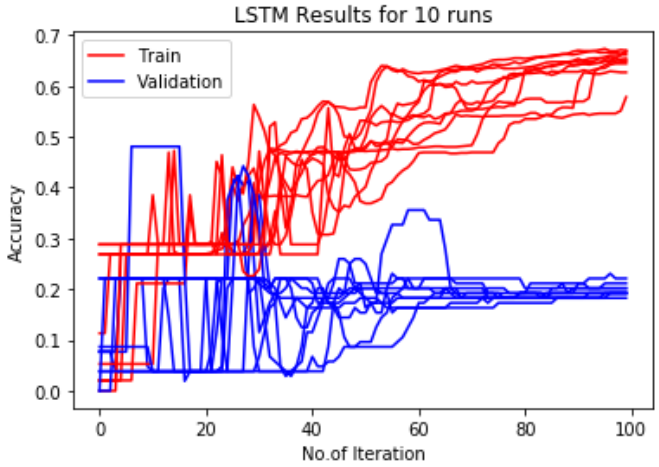

Fig. 4. Training and Validation Accuracy for LSTM Model

Given the weak results obtained from the LSTM, we subsequently moved on to a Random Forest model. This model achieves the best validation and test accuracies of 66% and 58% of all tried models, for $u = 5$. These results show that random forest model outperforms the LSTM model in our setting. To attempt further improvement of the accuracy, we also experimented with XGBoost [18]. The results from XGBoost are comparable with those of Random forest model, with validation accuracy and test accuracy of 65% and 60% respectively. Random forest was selected as the final task recognition model on account of its simplicity in terms of fewer tuning parameters, as compared to XGBoost.

Random Forest is also used for the **Task Predictor Model** and it achieves validation and test accuracies of 51% and 56% respectively for $v = 1$.

*5) **Model Deployment**:* The complete architecture after deployment is shown in figure 5. Without the MAGNET plugin, the user interacts with AF3 but there is no active guidance or feedback from AF3 (1). If SWTBOT is activated, it records the user interaction continuously in the background (2). The model is deployed as a task predictor plugin in Autofocus. The user clicks on the "Help Me" button and actions the trained machine learning model (3). The data from SWTBoT (4) is processed & analysed by the model to predict the user task(s) (5). The video hints according to the predicted task are displayed on the AF3 interface (6), thus providing active guidance for the user and completing the feedback loop.

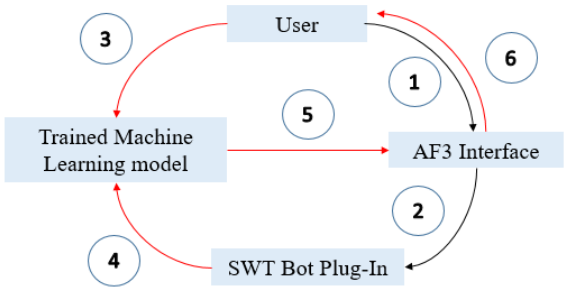

Fig. 5. Architecture of the Recommender System

IV. STATE OF THE ART

The published research that exists focuses on auto-completion mechanism for code-based IDEs, which differ from MDD tools in many ways, chiefly that IDEs for MDD typically deal with many levels of abstraction, often encoded as multiple graphical or textual DSLs. In these tools the process of building software involves mastering a set of different programming paradigms, as well as the inter-relations between different views of the system being developed.

The study of how developers use IDEs was initiated more than a decade ago, at the beginning of the 2000s. The Mylar framework [25] from Kersten and Murphy (later renamed to Mylyn) was one of the first of its kind to provide the developers with the means to define and follow tasks when building software. The framework, distributed as an Eclipse project, allows tracking low-level user commands to keep the state of the task up-to-date while dynamically adapting the IDE such that the navigation possibilities of the IDE are adapted to the developer's needs at each point of the development.

Several authors have concentrated specifically on gathering interaction information from IDEs. The work of Murphy on Eclipse for development in Java [30] was seminal, but more recent efforts exist such as the work from Amann*et al.* for Visual Studio. A survey of methods for collecting IDE interaction data has been proposed by Maalej *et al.* [29].

The natural step after collecting data is to analyse it for patterns that reveal developers' implicit development processes and habitudes, having as goal improving the usability of IDEs. Given the advent of advanced mining tools in the

2010s, authors such as Khodabandelou [26], Damevski [19] and Shepherd [20] have worked on what is now known as *process mining*. The authors have placed importance in not only coming up with models of developer behavior from low-level IDE logs of user-machine interaction, but also in automatically building models of those processes (e.g. using Markov chains) that can be understood by humans.

Since the past 5-10 years researchers have also started exploring how recommender systems can come to the help of software developers. Although the infamous "clippy" that shipped with early versions of the Microsoft office suite was a put off to having recommender systems as part of production software, advances in machine learning and human-computer interaction spawned new attempts at developing such functionality. Autodesk has implemented one such system for Autocad to improve the learnability of their tool by contextually proposing previously unseen commands. The company has then conducted a large study with more than 1000 users [27]. The authors of the study conclude that recommender systems are indeed useful and have a rich future in software applications. Damevski [27] as well as Bullmer [17] have explicitly explored classification algorithms to predict developer's behavior, much as we do. They report accuracies between 20-60%, which are in general lower than the accuracies we achieve. However, it is relevant to mention that the IDEs these authors explore are for code-based development, as opposed to our work on IDEs for model-driven development. In particular, their predictions are made on a large set of commands (e.g. 61 for the study in [17]) for which it will be naturally harder to reach high accuracies as for our work presented here (where 9 labels were used). The same study reports that neural networks have yielded a better accuracy, whereas in the study we present here they have performed the worst.

Outside academia, AF3 strongly relates to modern low-code tools, which employ model- and graphical-based software development principles to enable developers to quickly build, deploy and update applications. The two market leading tools at the time of the writing of this article are *mendix* [8] and the *OutSystems* [11] platform. *mendix* has implemented an AI-based system similar to MAGNET, where most likely steps are suggested to finish a logical workflow. Additionally, a "mentoring" mode is available to teach new developers how to build applications. *OutSystems* is currently investing very strongly in AI-based techniques to aid in software development, having created a laboratory [12] just for this purpose. As with AF3, the AI assistant of the *OutSystems* framework also predicts next steps in the development of the application, which *OutSystems* claims increases developer productivity by 25%. Note that both *mendix* and *OutSystems* are proprietary systems, while AutoFOCUS is distributed as open source.

JetBrains, the developers of a large range of code-based tools for software development, have recently started collecting data to improve their auto-completion systems since 2016 [3]. Additionally, a plugin for predictive coding [1] has been developed for the *Intellij* tool from JetBrains that uses machine learning for intelligent auto-completion, as well as for inserting snippets of code based on comments written in plain English.

V. Preliminary Results

As mentioned in section I, at fortiss we often offer workshops to demonstrate the various functionalities of AF3 to the new users. We have used on of such workshops to evaluate the usefulness of our IRS in one of the AF3 tutorial workshops. The focus group comprised 9 participants and was of mixed technical skills ranging from non-technical (e.g. managers and head of technical departments) to technical (e.g. software engineers). The existing format of the AF3 workshop is such that a human tutor to demonstrates the different functionalities of the AF3 at the beginning of the tutorial. The current workshop setting made a few of our recommendation videos redundant for the 3 participants having good programming skills. Nonetheless, the remaining 6 participants having less knowledge of programming (i.e., 5 managers and 1 professor) reported our system was helpful during the workshop giving their lower technical expertise. All feedback from the participants in the study was collected via a formal questionnaire.

As expected, we observed that providing information on AF3 to new users (i.e., technical or non-technical) reduces the use of our IRS. Seen from the reverse perspective, this points towards reducing human intervention in the tutorial (which is desired), which should lead to more usage of the IRS and a better evaluation of our proposal. It is worth mentioning that two participants (1 professor and one 1 software engineer) were very enthusiastic about MAGNET during the workshop, while the others were reticent to using it. This may indicate a polarized view on the usage of MAGNET, although such a claim calls for a future study.

In other interesting (and unexpected) feedback given to us during the workshop, it was suggested having to-the-point videos targeting the logical programming steps related to the tutorial exercise in the workshop. This is reminiscent of our earlier work on building process-aware modelling environments [28], where we proposed a DSL to express static development processes in MDD tools, to help for instance with meeting software certification rules in certain domains (e.g. avionics). Such feedback suggests fusing our previous work on process-aware modelling with the IRS we propose here to allow for both explicit and implicit (machine-learned) software development processes in AF3.

Another point worth discussing is the utilization of *SWTBot* [13] for the purpose of gathering the interaction data. The *SWTBot* tool has been built for performing the functional testing of the software interfaces. To the best of our knowledge has not been, up to now, utilized to gather interaction data for machine learning tasks.

VI. Future Directions

We have discussed an implementing a recommendation system using ML to ease the learnability of AF3, in particular in a tutorial setting. The results we present are preliminary but encourage us to further pursue this avenue of research. In

particular, are currently considering a number of follow-ups to this work:

- Improving data collection and labelling, as these processes have been up until now long and painstaking. We believe this can be partly achieved by building a "data acquisition" mode in AF3, where users can labels sets of their own interactions directly in the tool.
- Focusing specifically on improving user experience by studying how different hint delivery mechanisms (e.g. textual hints or highlighting certain parts of the IDE) improve user satisfaction. The Human-Centered Engineering group at fortiss is currently performing a study on the desirability these mechanisms in AF3. The results of such study should percolate into the MAGNET tool.
- Applying and evaluating different ML techniques to achieve more accurate next-task predictions. The study in this paper indicates that Random Forest achieves good performance – nonetheless these results should be taken with a grain of salt, given the low amount of labels (classes) that data acquisition has been done in the context of a tutorial. When broadening the scope of hints to the whole AF3 tool, it might very well be the case that other machine learning models will perform better.
- Studying how non user interaction data (e.g., AF3-model related data) can improve in task predictions for the new users. For the time being all predictions are based on user/widget interaction and no information about the state of the model of the embedded software system under construction is taken into consideration. Due to its size, it will is not possible to use the complete state of the model for either training or classification. But, by identifying deltas in the state of the model from one moment of time to the next and taking this information into consideration for training and prediction, we believe the accuracy of next task prediction will increase. Note that a system implementing the measurement of such deltas has already been implemented in AF3. We have indeed used this system to evaluate the state of a model in order to locate a developer in a pre-defined development process [28].
- Recalibrating the AF3 tutorial format to maximize the usage of the IRS during upcoming tutorial sessions.
- Extending the number of videos to include help for other parts of the AF3 tool, such as deployment, requirements engineering or formal verification. This is important not only to the IRS, but also as a means to improve the learnability of AF3 in general.

REFERENCES

[1] AI Predictive Coding plugin. https://plugins.jetbrains.com/plugin/9203-ai-predictive-coding/.

[2] AutoFOCUS3 Tutorials. https://af3.fortiss.org/docs/tutorials/.

[3] Data Collection Post for Intellij. https://blog.jetbrains.com/idea/tag/machine-learning/.

[4] Eclipse IDE. https://www.eclipse.org/ide/.

[5] Human-Centered Engineering. https://www.fortiss.org/en/research/projects/human-centered-engineering/.

[6] JetBrains. https://www.jetbrains.com/.

[7] MATLAB Simulink. https://www.mathworks.com/products/matlab.html.

[8] mendix. https://www.mendix.com/.

[9] MetaCase MetaEdit+. https://www.metacase.com/.

[10] National Instruments LabVIEW. http://www.ni.com/en-us/shop/labview/labview-details.html.

[11] OutSystems. https://www.outsystems.com/.

[12] outsystems.ai. https://www.outsystems.com/ai/.

[13] SWTBot. https://www.eclipse.org/swtbot/.

[14] V. Aravantinos, S. Voss, S. Teufl, F. Hölzl, and B. Schätz. Autofocus 3: Tooling concepts for seamless, model-based development of embedded systems. In *ACES-MB&WUCOR@MoDELS*, volume 1508 of *CEUR Workshop Proceedings*, pages 19–26. CEUR-WS.org, 2015.

[15] S. Barner, A. Diewald, J. Migge, A. Syed, G. Fohler, M. Faugère, and D. Gracia Pérez. DREAMS toolchain: Model-driven engineering of mixed-criticality systems. In *Proc. ACM/IEEE 20th Int.Conf. Model Driven Eng. Lang. Syst. (MODELS '17)*, pages 259–269. IEEE, 2017.

[16] W. Böhm, M. Junker, A. Vogelsang, S. Teufl, R. Pinger, and K. Rahn. A formal systems engineering approach in practice: An experience report. In *Proc. 1st Int. Workshop Software Engineering Research and Industrial Practices*, pages 34–41, New York, NY, USA, 2014. ACM.

[17] T. Bulmer, L. Montgomery, and D. Damian. Predicting developers' ide commands with machine learning. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pages 82–85, New York, NY, USA, 2018. ACM.

[18] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[19] K. Damevski, H. Chen, D. Shepherd, and L. Pollock. Interactive exploration of developer interaction traces using a hidden markov model. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR '16, pages 126–136, New York, NY, USA, 2016. ACM.

[20] K. Damevski, D. C. Shepherd, J. Schneider, and L. Pollock. Mining sequences of developer interactions in visual studio for usage smells. volume 43, pages 359–371, April 2017.

[21] J. de Lara and H. Vangheluwe. AToM3): A Tool for Multi-formalism and Meta-modelling. In *FASE*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer, 2002.

[22] J. Eder, S. Zverlov, S. Voss, M. Khalil, and A. Ipatiov. Bringing DSE to life: exploring the design space of an industrial automotive use case. In *Proc. ACM/IEEE 20th Int.Conf. Model Driven Eng. Lang. Syst. (MODELS '17)*, pages 270–280. IEEE, Sept. 2017.

[23] M. Feilkas, F. a. P. F. Hölzl, S. Rittmann, B. Schätz, W. Schwitzer, W. Sitou, M. Spichkova, and D. Trachtenherz. A refined top- down methodology for the development of automotive software systems: The keylessentry system case study. Technical Report TUM-I1103, Technische Universität München, 2011.

[24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[25] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ides. In M. Mezini and P. L. Tarr, editors, *AOSD*, pages 159–168. ACM, 2005.

[26] G. Khodabandelou, C. Hug, R. Deneckère, and C. Salinesi. Unsupervised discovery of intentional process models from event logs. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 282–291, New York, NY, USA, 2014. ACM.

[27] W. Li, J. Matejka, T. Grossman, and G. W. Fitzmaurice. Deploying communitycommands: A software command recommender system case study. *AI Magazine*, 36(3):19–34, 2015.

[28] L. Lúcio, S. bin Abid, S. Rahman, V. Aravantinos, R. Kuestner, and E. Harwardt. Process-aware model-driven development environments. In *MODELS (Satellite Events)*, volume 2019 of *CEUR Workshop Proceedings*, pages 405–411. CEUR-WS.org, 2017.

[29] W. Maalej, T. Fritz, and R. Robbes. Collecting and processing interaction data for recommendation systems. In *Recommendation Systems in Software Engineering*, pages 173–197. Springer, 2014.

[30] G. C. Murphy, M. Kersten, and L. Findlater. How are java software developers using the eclipse ide? *IEEE Softw.*, 23(4):76–83, July 2006.

[31] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.