Real-Time Personalization in Adaptive IDEs

Matthias Schmidmaier*
schmidmaier@fortiss.org
fortiss GmbH
Munich, Germany

Zhiwei Han*
han@fortiss.org
fortiss GmbH
Munich, Germany

Thomas Weber*
weber@fortiss.org
fortiss GmbH
Munich, Germany

Yuanting Liu
liu@fortiss.org
fortiss GmbH
Munich, Germany

Heinrich Hußmann
hussmann@ifi.lmu.de
LMU Munich
Munich, Germany

ABSTRACT

Integrated Development Environments (IDEs) are used for a variety of software development tasks. Their complexity makes them challenging to use though, especially for less experienced developers. In this paper, we outline our approach for an user-adaptive IDE that is able to track the interactions, recognize the user's intent and expertise, and provide relevant, personalized recommendations in real-time. To obtain a user model and provide recommendations, interaction data is processed in a two-stage process: first, we derive a bandit based global model of general task patterns from a dataset of labeled interactions. Second, when the user is working with the IDE, we apply a pre-trained classifier in real-time to get task labels from the user's interactions. With those and user feedback we fine-tune a local copy of the global model. As a result, we obtain a personalized user model which provides user-specific recommendations. We finally present various approaches for using these recommendations to adapt the IDE's interface. Modifications range from visual highlighting to task automation, including explanatory feedback.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; **User models**; **Graphical user interfaces**; • **Computing methodologies** → **Reinforcement learning**.

KEYWORDS

Adaptive IDE; User Modeling; Personalized Recommendendation Systems; Human-Centered Machine Learning;

ACM Reference Format:
Matthias Schmidmaier, Zhiwei Han, Thomas Weber, Yuanting Liu, and Heinrich Hußmann. 2019. Real-Time Personalization in Adaptive IDEs. In *27th Conference on User Modeling, Adaptation and Personalization Adjunct (UMAP'19 Adjunct), June 9–12, 2019, Larnaca, Cyprus.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3314183.3324975

*Authors contributed equally to this research.

UMAP'19 Adjunct, June 9–12, 2019, Larnaca, Cyprus

ACM ISBN 978-1-4503-6711-0/19/06…$15.00
https://doi.org/10.1145/3314183.3324975

1 INTRODUCTION

IDEs are software development tools with an abundance of functionality but consequently also with the tendency to overwhelm users with their complexity. Command-search and context-based hints are some of the methods which modern IDEs use to address this issue. Current IDEs also support various user adaptations, enabling the developer to adjust his work environment to his personal preferences. One step further to these adaptable systems are adaptive IDEs, that apply user modeling to automatically adjust and personalize the system to a developer's individual preferences and demands. While the concept of user modeling and adaptive systems is well-established in the web context, for example for content recommendation, applications in software development are still scarce. Previous work in the field mostly addresses identification and prediction of user behavior based on joint interaction data collected from multiple developers. By contrast, our approach also provides personalized recommendations and interface adaptations. For this purpose, we introduce a two-tier, bandit-based approach for real-time user modeling (Sect. 3) as well as approaches for how to use predictions for interface adaptation (Sect. 4). In addition, we take system transparency and user control into account.

2 RELATED WORK

Several publications investigated concepts for user-adaptation in software development. Robillard et al. [16] present an overview on recommendation systems for software engineering (RSSE). They describe systems that provide recommendations based on user characteristics, conducted task, task characteristics or past user actions. The presented applications support developers by recommending reuse of specific code fragments, expert consultants, code examples, information navigation or what parts of code to change next. A similar overview on software development recommendation systems is given by Happel et al. [10].

Murphy-Hill et al. [15] combine collaborative filtering and multiple discovery algorithms to make developer command recommendations. Based on a dataset of around 4000 developers, they were able to recommend commands with an accuracy of about 30%. In addition, they conducted a live study, manually presenting recommendations to novice and expert users to evaluate usefulness and acceptance. In a more recent work, Damevski et al. [5] present a topic modeling approach (Temporal Latent Dirichlet Allocation) for predicting future developer behavior in an IDE. However, they do not implement or evaluate any forms of recommendation or system

adaptation. Similarly, Bulmer et al. [2] also predict developers' next interaction with up to 64% accuracy using a neural network trained on a dataset with interaction data from 3000 developers.

All of these approaches make predictions by training on offline datasets. While we follow the same strategy in our first tier, in our approach we add a second level of personalization by continuously readjusting the model based on real-time user interaction. Regarding personalized recommendation in IDEs, Gasparic et al. [9] introduced a command recommender system based on the developer's current work context and his previous knowledge of commands. With this approach, the user also gets personalized recommendations for relevant commands that he is assumingly not aware of.

3 USER MODELING

In this section, we propose a bandit model approach for user modeling in adaptive IDEs with two tiers, *Initialization* and *Personalization*. After describing the two levels of interaction data granularity, we show how this data is used in the two tiers. The following sections describe in detail how to construct the user model and the recommender system, which are the foundation for adapting the IDE's interface to the user's needs and preferences.

3.1 Data Description

To build an adaptive IDE we closely collaborated with the developers of AutoFocus3 (AF3) [1], an IDE for model-based development. This collaboration allowed us to record user interaction in great detail, directly inside the IDE. Input at the click- or mouse-move-level is noisy though, for example due to random user exploration and meaningless interactions. This makes it hard to identify task patterns and user preferences. So, following an approach described in [11] and [17], we introduce an additional level of granularity, differentiating between fine-grained, low-level interactions as recorded by the IDE and high-level task labels. Each task is a labeled sequence of low-level interactions. Selecting adequate task labels is domain-specific and should be chosen with feedback from domain experts.

3.2 A Two-Tier Approach to Personalization

To get value from our data we propose a two-tier bandit model framework. In the initialization-tier, illustrated in the upper part of Fig. 1, we first train an interaction classifier (see Sect. 3.2.1) and derive user expertise criteria (see Sect. 3.2.2) from the training data. Then, we initialize a global user model (see Sect. 3.3), extracting from the training data how often and in which order different tasks are performed (task frequencies). During this initialization stage, only offline training data is required. A copy of the global model serves as base for the personalization-tier, which in contrast to the first tier also requires real-time user interaction. As shown in the lower part of Fig. 1, the classifier created in the first tier labels the low-level real-time data to obtain the high-level task description.

As next step, the user expertise level is determined by comparing the local task patterns with the global task patterns regarding the expertise criteria (see Sect. 3.2.2). Subsequently, an individual local user model is obtained by incrementally fine-tuning the local copy of the global model with the real-time high-level data and implicit user feedback. In this way, during fine-tuning the local model is adapted to the user's individual task patterns, leading to a personalized copy of the global task patterns.

There are mainly three advantages of applying this two-tier approach. First, we can transfer the prior knowledge (general user task patterns) directly into personalized recommendation, so that we do not suffer from the common cold-start problem. Second, since the proposed framework allows us to incrementally update the local user model, we only fine tune with the user's interactions and do not have to re-train the model from scratch. Third, with data of individual users only captured and processed locally, our approach offers a high level of privacy and data sovereignty (see also Sect. 4.3). In the following, we introduce the previously described components in detail.

3.2.1 Interaction Classifier. While the training data is labeled with task descriptors, the real-time interaction data stream is not. However, our user models and bandit based recommendation models (see Sect. 3.3 and 3.3.1 respectively) are based on the high-level task descriptors. Thus, we train an interaction classifier that maps low-level interaction data to high-level task descriptions. The classifier is trained in a supervised manner by minimizing the high-level task description classification error on the training data. As an example, assume there is a low-level interaction trace $l = \{b, a, c, d, e, f\}$ with high-level task descriptions $h = \{A, A, A, A, B, B\}$ in the training data. In training, the interaction classifier takes N (here $N = 3$) consecutive low-level interactions $\{b, a, c\}$ as the input and learns to predict the first corresponding high-level task description A. The benefit of this method is that it can capture the temporal dependencies maintained in the low-level interactions.

3.2.2 Expertise Criteria. A high-level task can be described through a pattern, consisting of a chronological sequence of low-level interactions. By comparing a individual user's task patterns to the common task patterns, a user's level of expertise can be estimated (see knowledge modeling in [1]). For expertise level determination, we assume that advanced users need fewer actions for finishing the same tasks than less experienced users. Consequently, for all user levels, we set a threshold of average number of steps for finishing tasks. The average step number of a user is no more than the threshold of his expertise level.

3.3 User Model

In context of IDE task recommendation, we define a user model $\mathbf{M}$ as a function that maps a user representation u and a high-level task representation e to a real value. This real value is the payoff of choosing e given u and can also be interpreted as the user preference to the task e of the user u. More formally, it can be defined as:

$$\mathbf{M} : \mathcal{U} \times \mathcal{E} \mapsto \mathbb{R}, \tag{1}$$

where $\mathcal{U}$ is the user representation space and $\mathcal{E}$ is the high-level task representation space. There can be different choices of mathematical models for $\mathbf{M}$, for example regression model or neural networks. Since for our purposes we investigate only a limited number of tasks and user states, we can use a tabular form of user model.

Its row index is user representation and column index is high-level task representation. We denote task ID as the high-level task

[1] https://af3.fortiss.org

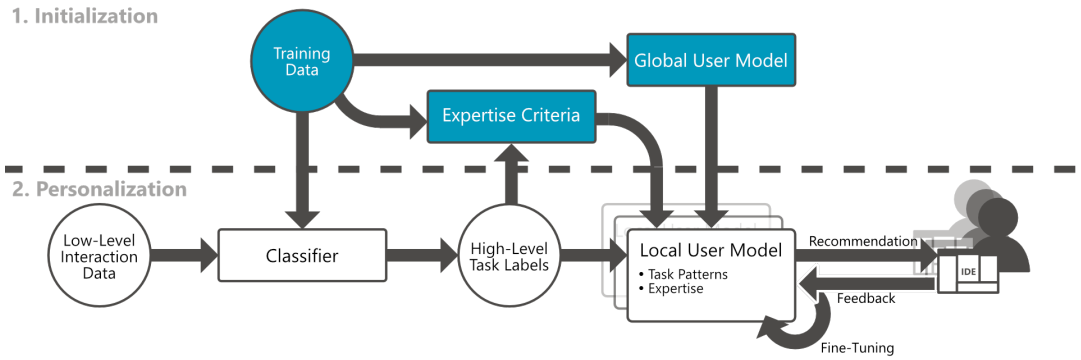

Figure 1: Two-tier user modeling and recommendation approach.

	A	B	C
ABC	0.7	0.1	0.2

Table 1: Exemplary row in the tabular user model

Three tasks *A*, *B*, *C*, following the trigram *ABC* in the training with 7, 1, 2 occurrences respectively.

representation. The entry value of this table is the estimated payoff in context of bandit problems. For user representations, we use a *N*-Gram model [4] based representation, which originate from the field of language modeling. In our *N*-Gram setup, user representation is formalized by concatenating *N* last task representation in one aggregation as defined in the following equation,

$$u_t = [e_{u,t-N} \dots e_{u,t-2} \;\; e_{u,t-1}], \tag{2}$$

where u_t is the representation of user u at time t, $e_{u,t}$ is the task taken by user u at time t.

Throughout this work, we have two types of user models, which we both consider as tables. One is the global user model which captures the common user task patterns like task types and order of task execution. Each row of the global model is initialized with the task frequencies of the corresponding user representation as shown in Tab. 1. Since we do not get real-time frequencies of high-level tasks, training of local user model should follow the typical contextual bandit model approach, which will be introduced in the following section.

3.3.1 Recommendation Algorithm. We model personalized IDE task recommendation as a contextual multi-armed bandit problem [13, 14]. We consider all available tasks to be recommended as arms, user representation and task representation as context. Therefore, a contextual-bandit based recommender *R* evaluates all possible tasks with learned user models. Intuitively, the recommendation is made by taking the arm with the maximal payoff. The update rule of the recommender *R* can be concluded as following, at each time t:

(1) The recommender *R* observes a new coming user u and a set of n available tasks E_t. *R* summarizes the information of u and E_t into n feature vectors, which can also be referred as arm contexts.
(2) *R* forwards the arm contextual information to the corresponding user model and gets the estimated payoffs of all arms. It selects a task e_t from E_t with maximal estimated payoff.
(3) *R* receives the corresponding user feedback (payoff) r_t of e_t and updates the selection strategy.

It is necessary to emphasize that the payoff r_t here only depends on the user representation u and selected task e_t. The unchosen tasks don't affect the payoff r_t. When a recommended task is selected by a user, a payoff of 1 is incurred; otherwise, the payoff is -1. It is equivalent to maximizing the recommendation accuracy, which in turn is the same as maximizing the total expected payoff in our bandit model. Differing from the traditional multi-armed bandit approach, we don't further explore other actions in the global model. That is because the global model was trained in a supervised manner and can well represent the general user behaviors. The candidates of the optimal action are reasonably among the actions with the highest payoffs. Therefore, inefficient exploration will even reduce the algorithm performance and user satisfaction.

3.4 Proof of Concept

Although we developed this approach within a project of a published, open-source modeling IDE, development and distributions processes do not allow for proper validation regarding amount and quality of captured interaction data at this time. So, in order to get a first proof of concept, we create synthetic data representing individual high-level task patterns as described in Tab. 1.

3.4.1 Synthetic Dataset. In a first step, we generate a synthetic dataset to train a tier-one global model. We initially define $N_{pat} = 10$ ground truth task patterns consisting of randomly generated sequences of five to ten high-level tasks. To simulate diverse real-world interaction events, we define $N_{task} = 10$ available high-level tasks. The generated task patterns then are used to create $N_{train} = 10000$ synthetic user records. Each record is a set of up to 50 randomly chosen task patterns. In this process, individuality of each user is introduced by applying a corruption function to the ground truth task patterns. Here, we interpret individual user behavior as differing preferences in interaction order for the same task. Therefore, the applied corruption strategy is to randomly swap high-level tasks in individual user records. As a result, we obtain a dataset of individual task patterns, that we use to train a global tabular user model. In the same way, we create a second dataset of

size $N_{val} = 1000$, that we use for model validation. In a further step, we simulate real-time user feedback in order to gain a personalized local user model. Thereby user feedback on recommendations is simulated in a simplified way: if the predicted recommendation hits the user action, a positive feedback is given. Otherwise, the local model receives a negative feedback.

3.4.2 Model Evaluation. For accuracy evaluation, we create multiple synthetic datasets, modifying the parameters N_{task} and N_{pat} to train multiple global models. Fig. 2 illustrates accuracy for global and local model resulting of ten repetitions with a growing number of available high-level tasks $N_{task} = 1..10$. Correspondingly, Fig. 3 illustrates the accuracy of ten repetitions with a growing number of defined task patterns $N_{pat} = 1..10$. According to these results, our two-tier model approach continuously outperforms the single global model gaining a performance improvement over 5%. With a parameter setting that roughly depicts interaction patterns observed in the real-world modeling IDE AutoFocus3 ($N_{task} = 8$, $N_{pat} = 10$), the local model accuracy is around 36%, providing an improvement of around 6% compared to the global model.

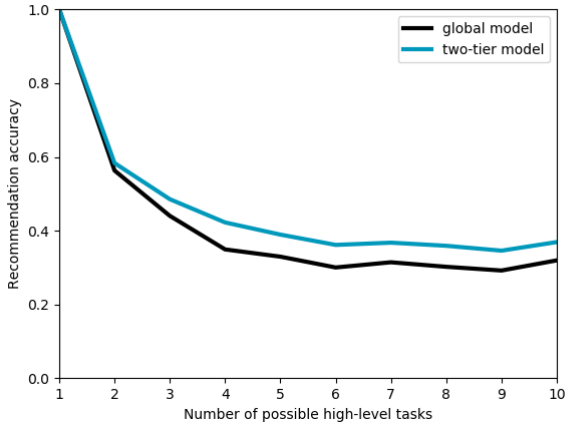

Figure 2: Model accuracy over $N_{task} = 1..10$ with $N_{pat} = 10$.

3.4.3 Discussion. We are aware that the use of a synthetic dataset is linked to various limitations. So for example real-world noise in interaction behavior is hard to simulate. However, results suggest that our two-tier approach in general is feasible and promising regarding the accuracy improvement even given very sparse user feedback for the local model. Using real interaction data, we assume that accuracy improvement may be even larger due to possible user behavior exploration and potentially much more dense user feedback for the local model adaptation. As further described in Sect. 5, we plan to implement our approach using real IDE interaction datasets, possibly following another run on synthetic data that is closer to real IDE interaction than our current randomly generated data. In addition, we may compare alternative mathematical models, taking evaluations of related approaches into account.

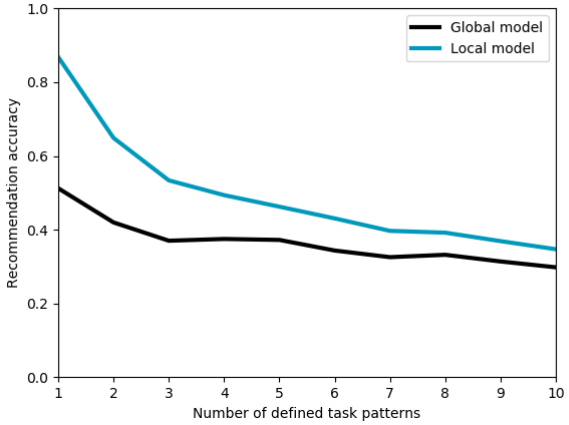

Figure 3: Model accuracy over $N_{pat} = 1..10$ with $N_{task} = 10$.

4 SYSTEM ADAPTATION

In the previous section we described how to identify and predict individual user behavior in real-time. These predicted task patterns can now be used to perform personalized system adaptations, that means adapt the user interface according to the personalized recommendations. Based on the classification described by Bunt et al. [3], adaptations can be divided into the following categories: *content*, *presentation*, and *modality adaptation*. Referring to Jameson [12], we add an additional category *functionality adaptation*, achieved for example by automating routine tasks or a adapting a system's dialog strategy. In the following sections we describe our approaches on how to apply the user model and the recommendations for adapting the IDE's interface.

4.1 Functionality Adaptation

As described in Sect. 3.1, a local user model describes the task patterns of an individual developer. These patterns can be used to dynamically support the user's workflow according to his intentions and preferences. That means that when performing a specific task, the IDE can support the execution of subsequent tasks with appropriate adaptions. The following sections describe different forms of functionality adaptation for this kind of individual workflow optimization.

4.1.1 Action Shortcut. One variant of functionality adaptation is to provide to the user a shortcut to the predicted next action. Such a *quick access* can be presented e.g. with a permanently accessible "quick-fix"-button. For multiple recommendations, a temporary displayed radial menu can provide quick access (see Fig. 4). In addition, a hotkey can be assigned to execute the next recommended action. With respect to transparency and usability, all of these options - above all the last one - have the issue that it can be unclear why an action was recommended, so it should come with some form of explanation (see Sect. 4.3).

4.1.2 Action Initialization. Beyond offering quick access, a further functionality adaption is to automatically initiate the predicted

action. This can be accomplished by for example focusing relevant interface elements like windows, buttons or text input fields. For example when a possible high-level task *renaming* is predicted, the system can initiate the action by opening the renaming dialog. Again, these functionality should provide some kind of explanation to the user, especially when changing the input or cursor focus.

4.1.3 Action Automation. The most thorough approach would be to not just initiate an action, but automate the complete execution of the predicted next step. An adaptive IDE could for example automatically compile and run a project at points where this has been identified as part of the user's routine. However, as this is a deep intervention in the user's workflow, it requires correspondingly designed feedback and undo functionality.

4.2 Presentation Adaptation

Contrary to functionality adaptions, presentation adaptations alter the interface in order to show or hide selected content and is often used in adaptive-web context [3].

4.2.1 Expertise-Level Based Hints. As described in Sect. 3, the local user model is able to estimate the level of expertise of an individual user. That means, if a developer is performing a specific task for the first time, we can provide him with a detailed task description instead of action shortcuts. For more experienced users, the hint would remain hidden. In this way, developers get the possibility to learn new tasks while experienced users are not distracted or annoyed. Furthermore, hints could also provide experienced users with additional information about the recommended task. These hints can be realized in different ways, for example as short how-to animations as sketched in Fig. 4, textual notifications or by highlighting relevant interface elements.

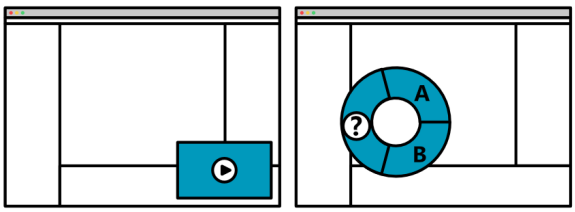

Figure 4: Interface adaptions.
Video instructions (left) and radial menu (right) with quick access to recommended actions A, B and explanation.

4.2.2 Information Dimming. While hints represent a way of highlighting and showing additional information, another form of adaptation is to remove less relevant information [3]. This can be achieved by dynamically hiding or minimizing interface elements to reduce cognitive overload and support task focus flow. In an IDE, elements to hide are for example the individual panels that are not relevant for the steps in the predicted workflow and based on the users preferences: a developer might prefer a maximized console panel or file overview for one task while for another task he prefers a maximized code window. Based on this information, other panels can be minimized, scaled or just made visually less intrusive, for example via lighter colors. Especially re-arranging or minimizing interface elements can easily be confusing for the user though. So mechanism have to be in place to ensure the decision and adaption process to be transparent to the user. Furthermore the user must maintain control of the interface to undo adaptions, should they turn out to be undesired.

4.3 Transparency and Control

Ensuring the feeling of transparency and control is an important factor in adaptive systems [6, 8]. A transparent system should always be able to answer: why has the functionality or layout changed and how can I (the user) modify or undo these adaptations?

4.3.1 Explanation Access. To enable continuous access, we argue that explanations about system adaptivity should be permanently available, for example in the IDE's settings. However, additional direct access to explanation could increase the user's understanding of why adaptations occur. One possible implementation is to provide an interactive element that leads to further information on click. This could be a permanent button which is highlighted on adaptation, or an element that is faded in when an adaptation occurs, as shown in Fig. 4. Depending on the informational content, an explanation could also be directly shown on system adaptation, for example in a dedicated explanation panel or with short textual or graphical overlays.

4.3.2 Explanation Content. Another important factor to be considered is the explanation's information content and granularity. We suggest to initially provide explanations with a minimum level of detail and to enable some kind of zoom-into-details functionality. So for example regarding the task expertise level (see Sect. 3.2.2), the developer would be presented some kind of progress visualization which shows the currently assigned level of expertise. On request, a more detailed explanation could inform the user how certain levels of expertise lead to certain adaptations or recommendations. Finally, an even more detailed explanation screen could explain how level of explanation is determined.

4.3.3 User Control. As described in Sect. 3, the local user model is fine tuned through implicit user feedback, for example when the user follows or ignores given recommendations. However, the user should also be able to explicitly influence the system's adaptive behavior [7, 8]. Correcting individual adaptive actions can easily be done by observing common interface elements like the undo button or hotkey. More complex modifications of the adaptation behavior will most likely require some dedicated settings menu.

4.3.4 Privacy and Data Protection. Finally, we want to mention that in every system processing user data, privacy and transparency of the data processing should be considered. Besides adding corresponding information to the IDE's terms, a more accommodating approach would be to add information about captured and processed data to the adaptation feedback. This can include an interface element which visualizes when and which data is captured as well as when and where it is processed and stored.

5 CONCLUSION AND OUTLOOK

In this paper we have outlined our approach for an adaptive IDE, how to make personalized interaction predictions and recommendations and how to adapt the IDE interface accordingly. Our two-tiered approach consists of a common bandit based global model which is refined into a local personalized one for each user, using real-time classified interaction data. As a first proof of concept, we implemented this approach using synthetic data. The results regarding accuracy suggest the viability of our two-tiered approach. In addition, this methodology offers benefits regarding offline model training and privacy. We have also described a number of potential interface adaptions, taking into account system transparency and user control.

As mentioned in Sect. 3.1, the described concepts have been developed for the open-source model-based IDE AutoFocus3. Our next goal is to apply our approach to IDEs with a larger user base, giving us access to a larger amount of interaction data. There are also some further domains, like game development with for example *Unity3D (https://unity3d.com)*, where a complex development tool is used by a diverse group of users that can benefit greatly from a personalized and skill-based adaptive interface. In addition, we will evaluate the different proposed adaptions with respect to usability and how beneficial they are for providing recommendations.

ACKNOWLEDGMENTS

This research has been funded by the *Bavarian Ministry of Economic Affairs, Regional Development and Energy* as part of project MAGNET (EF 2018_006).

REFERENCES

[1] Peter Brusilovsky and Eva Millán. 2007. User Models for Adaptive Hypermedia and Adaptive Educational Systems. In *The Adaptive Web: Methods and Strategies of Web Personalization*, Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 3–53.

[2] Tyson Bulmer, Lloyd Montgomery, and Daniela Damian. 2018. Predicting Developers' IDE Commands with Machine Learning. (2018).

[3] Andrea Bunt, Giuseppe Carenini, and Cristina Conati. 2007. Adaptive Content Presentation for the Web. In *The Adaptive Web*, Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl (Eds.). Springer-Verlag, Berlin, Heidelberg, Chapter Adaptive Content Presentation for the Web, 409–432.

[4] Stanley F Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language* 13, 4 (1999), 359–394.

[5] K Damevski, H Chen, D Shepherd, N A Kraft, and L Pollock. 2018. Predicting Future Developer Behavior in the IDE Using Topic Models. *IEEE Trans. Software Eng.* (2018), 1–1.

[6] Malin Eiband, Hanna Schneider, Mark Bilandzic, Julian Fazekas-Con, Mareike Haug, and Heinrich Hussmann. 2018. Bringing Transparency Design into Practice. In *23rd International Conference on Intelligent User Interfaces (IUI '18)*. ACM, New York, NY, USA, 211–223.

[7] Christoph Evers, Romy Kniewel, Kurt Geihs, and Ludger Schmidt. 2014. The user in the loop: Enabling user participation for self-adaptive applications. *Future Gener. Comput. Syst.* 34 (May 2014), 110–123.

[8] Gerhard Fischer. 2001. User Modeling in Human Computer Interaction. *User Model. User-adapt Interact.* 11 (2001), 65–86.

[9] Marko Gasparic, Tural Gurbanov, and Francesco Ricci. 2017. Context-aware Integrated Development Environment Command Recommender Systems. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2017)*. IEEE Press, Piscataway, NJ, USA, 688–693.

[10] Hans-Jörg Happel and Walid Maalej. 2008. Potentials and Challenges of Recommendation Systems for Software Development. In *Proceedings of the 2008 International Workshop on Recommendation Systems for Software Engineering (RSSE '08)*. ACM, New York, NY, USA, 11–15.

[11] Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse. 1998. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. 256–265.

[12] Anthony Jameson. 2009. Adaptive Interfaces and Agents: Design Issues, Solutions, and Applications. In *Human-Computer Interaction*, Andrew Sears and Julie Jacko (Eds.). Human Factors and Ergonomics, Vol. 20093960. CRC Press, 105–130.

[13] John Langford and Tong Zhang. 2008. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in neural information processing systems*. 817–824.

[14] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 661–670.

[15] Emerson Murphy-Hill, Rahul Jiresal, and Gail C Murphy. 2012. Improving Software Developers' Fluency by Recommending Development Environment Commands. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE '12)*. ACM, New York, NY, USA, 42:1–42:11.

[16] M Robillard, R Walker, and T Zimmermann. 2010. Recommendation Systems for Software Engineering. *IEEE Softw.* 27, 4 (July 2010), 80–86.

[17] John Wilkie, Ziad Al Halabi, Alperen Karaoglu, Jiafeng Liao, George Ndungu, Chaiyong Ragkhitwetsagul, Matheus Paixao, and Jens Krinke. 2018. Who's this?: developer identification using IDE event data. In *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 90–93.