Modeling Context-Aware Command Recommendation and Acceptance in an IDE

Marko Gasparic, Francesco Ricci
Free University of Bozen-Bolzano
Piazza Domenicani, 3, I-39100 Bozen-Bolzano, Italy
Email: marko.gasparic@stud-inf.unibz.it, fricci@unibz.it

***Abstract*—For software developers to use the full range of available commands in an integrated development environment, one has to provide proactive support which can suggest unknown commands that could be useful for the task at hand. Researchers started exploring the potential of recommender systems to provide this type of help, but so far there are still very few contributions. We propose a new multi-criteria context-aware rating prediction model that can be used to predict the user choice of either to accept or reject an IDE command recommendation. Individual command recommendation evaluation criteria are: performance expectancy, effort expectancy, and social influence; besides, the overall evaluation/rating is the intention to use a command. We have identified four types of contexts, namely, current practice, environment, interaction, and recommendation presentation context. The model is aimed at improving recommendation quality and enabling more effective recommendation presentations.**

***Index Terms*—Recommender system, model, context-aware, multi-criteria, command.**

I. Introduction

High-functionality applications (HFAs) are complex systems which serve the needs of a large and diverse user population [1]. Since most of their users are not experts, and they only want to efficiently accomplish the task at hand, they are often not interested in exploring and using *novel* features that actually could improve their job performance. Integrated development environments (IDEs) are HFAs, and Murphy-Hill [2] found that software developers are using only a small subset of commands (average Eclipse user is using less than 50 commands out of 1100 that are available).

Unless some proactive mechanism, which suggests specific IDE commands that the software developer does not know, is introduced, she will not learn *new* useful commands. On the other hand, the system should not inform the user about commands that are irrelevant for her job and it should not recommend already known commands; i.e., "Did You Know" and "Tip of the Day" recommendations do not suffice [1].

Recommender systems (RSs) are personalised information search and filtering tools that suggest useful items [3]; they have been primarily used in eCommerce, for recommending books, CDs, or movies. Nowadays, they are applied in many domains, also in software engineering [4].

Researchers are exploring usage of RSs as active help systems too, but while there are still very few contributions ([5], [6], [7], and [8]), it seems that the effectiveness of current command recommenders can be improved further.

The goal of a RS is to recommend an item with the largest utility for the user. According to [2], only 12% of developers that submitted Eclipse usage data used *open resource* shortcut, which experts classify as very useful. An effective command RS should recommend a shortcut when it can be used instantly, only to the developers who are not familiar with it, and the RS should tailor the argument supporting the recommendation to the developer characteristics.

Existing command RSs are based on "most popular" or collaborative filtering techniques. In this article we will posit that multi-criteria context-aware RSs could generate more convincing and useful command recommendations. Here we do not propose new algorithms, since we assume that existing (rating prediction) algorithms are of sufficiently high quality and can be applied also in this domain. We want to focus on the analysis of the problem, discussing the application scenario, the type of recommendations required and how the presentation of the recommendations should be adapted to the context and the user, and in particular the choice model of the user, which specifies how the user of an IDE evaluates the choice of either to accept or reject a command recommendation.

In this article we present a new rating prediction model for command RSs, which is aimed at improving recommendation quality and enabling more effective recommendation presentations. In section II we present preexisting models and theories that influenced the design of our model, which is described in section III. In the last section we outline our plan for the future work.

II. Related Work

A. Recommender Systems

RSs are helping their users to make better choices and are very common in Internet applications nowadays. They exploit data mining and information retrieval techniques to predict which items are best suited for the user's needs and then they recommend them [3].

More traditional RSs do not take into account any contextual information, even though ignoring the context of a recommendation can jeopardize the relevance of the recommendations in many domains [9]. For instance, if programmer's actions in the development environment are not taken into account, the RS could recommend commands that she already knows or commands that are not applicable. The rating function in

978-1-4673-7037-0/15 $31.00 © 2015 IEEE
DOI 10.1109/CSD.2015.7

context-free RSs is of the type $User \times Item \rightarrow Rating$, which means that the user's rating for an item is determined only by the user and item properties (certain users like certain items). In context-aware RSs (CARSs) the rating function is of the type $User \times Item \times Context \rightarrow Rating$, which means that the user's rating is affected also by some contextual factors. Albeit CARSs take into account more potentially relevant data for generating recommendations than traditional systems, they still model utility of an item as a single value.

Adomavicius et al. claim that capturing more fine grained preferences can improve recommendation quality [10]. For example, it is reasonable to believe that a command recommendation system would provide better recommendations if it could predict the perceived usefulness and perceived ease-of-use, compared to the system that would only capture overall rating of a command. Multi-criteria CARSs can be modeled with the following function type:

$$User \times Item \times Context \rightarrow R_o \times R_1 \times \ldots \times R_k \qquad (1)$$

where $R_1 \times \ldots \times R_k$ represent the ratings for individual criteria and R_o represents the overall rating[1]. In our multi-criteria rating prediction model, performance expectancy, effort expectancy, and social influence ratings are individual rating criteria and the intention to use a command is the overall rating.

B. Functionality and User Acceptance

Fischer [1], in relation to user's knowledge, splits the full functionality of a system in four groups:

- D1: concepts that are well known, easily applied, and used regularly,
- D2: concepts that are vaguely known and used only occasionally,
- D3: concepts that users believe to exist,
- D4: functionality provided by the system.

To take advantage of the functionality contained in D1 the users do not need any help. To use D2 functionality, passive help systems are usually needed, e.g., user manual, and they can also help users to explore D3 functionality, however, passive help systems are not sufficient to discover D4 functionality that is not included in D3.

In figure 1, the T region represents the functionality that is relevant to the task at hand. The goal of active help systems is to point out *only* the functionality that does help the user to accomplish her task [1].

If we want to recommend commands that the user will use, we have to understand what affects user's intention to use them. Unified Theory of Acceptance and Use of Technology (UTAUT) [11] helps to understand the drivers of technology acceptance. It integrates elements of eight competing models, which were outperformed by UTAUT in explaining variance in user intentions to use information technology [11].

[1]Not all multi-criteria RSs include the overall rating in the rating prediction model [10].

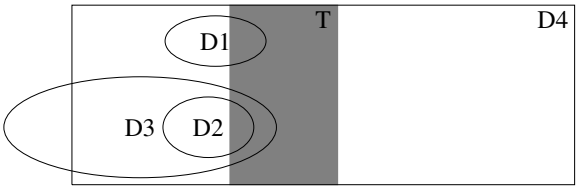

Fig. 1. Functionality and its relevance to the task at hand [1]

According to [11], three determinants directly affect intention to use a new system: performance expectancy, effort expectancy, and social influence. Performance expectancy is the strength of belief that the usage of the new system will improve job performance. Effort expectancy defines user's perception of simplicity of use of the new system. Social influence are opinions of important others[2].

C. Human Decision Making

Jameson [12] devised ASPECT model, which describes everyday human decision making that is relevant also in the context of RSs. He assumes that the various possible choice patterns that choosers adopt are of six fundamental types:

- *Attribute-Based Choice*: each recommended item is viewed as an object that can be described with evaluation-relevant attributes, and (relative) desirability of an item can be estimated in terms of evaluations of relevant attributes as well; it is important to note that a user will cognitively reduce the set of relevant attributes to a manageable size if it is too large.
- *Consequence-Based Choice*: the user is reflecting on (some) anticipated consequences of the items and selects the one that would in her opinion perform (relatively) well.
- *Experience-based Choice*: the user applies recognition-primed decision making[3] or affect heuristics, acts on the basis of a habit, or chooses previously reinforced response; this pattern is only applicable if the user has made similar choices in the past.
- *Socially-Based Choice*: when user can estimate decisions of other relevant people she is guided by their examples, expectations, or advice.
- *Policy-Based Choice*: the user identifies the preferred option based on the policy applicable in the current context; this pattern is only applicable if the user encounters similar choices on regular basis.
- *Trial and Error-Based Choice*: the user (maybe randomly) selects one item and observes the consequences, if she is not satisfied with the outcome she makes a new selection, taking into account also the new knowledge.

We note that these choice patterns are often combined in decision process [12].

[2]It is not necessary that those opinions were expressed explicitly, also user's estimations of opinions of others affect the intention to use the system.

[3]Exposure to some stimulus increases the accessibility of information already existing in memory and influences the response [13].

D. Existing Command Recommender Systems

Linton and Schaefer [5] designed a RS that monitors command usage within an organization, calculates average usages, and recommends to the user to use a command more (or less) often, if she was using it less (or more) often than the average member of organization. To the best of our knowledge, it is the first command RS that provided personalized recommendations. However, it is questionable how effective this type of recommendations is, since the RS assumes that all the users (within the organisation) should use the tool similarly, which is not the purpose of HFAs. In our case, recommending to a tester to use less often *run as* command and more often *rename selection* command is not sensible.

Matejka et al. [6] applied collaborative filtering algorithms and rule-based domain knowledge to provide command recommendations in AutoCAD. Murphy-Hill et al. [7] applied the same algorithms as [6] and four new algorithms to recommend commands in Eclipse. They showed that it is feasible to recommend useful IDE commands to software developers; in the offline evaluation the best algorithm was accurate in 30% of the cases, while in the online evaluation 26% of commands recommended to experts were novel and 80% were novel to novices.

In the next section we present a new rating prediction model, which is multi-criteria and context-aware. The inclusion of context was inspired also by Spyglass RS [8], that is an active help system and can recommend 3 tools in the Rational Team Concert environment. Developers who participated in a laboratory study and used Spyglass became aware of navigational tools as well as those who read tutorial, but with less up-front effort. Moreover, contextual recommendations improved navigation efficiency compared to tutorials. We would like to improve developers' awareness and efficiency by recommending commands of any type, as long as they are applicable in the developer's context.

III. Model

In this section we describe our multi-criteria context-aware model for command recommendation. The model accounts for factors that influence software developer's decisions when an IDE command is recommended. We focus only on recommendations for one command, since, even though a RS can concurrently recommend several commands, the decision whether to execute a specific command (instantly or in the future) is not directly influenced by other recommendations.

UTAUT model can assess the likelihood of success of a new technology. We applied it to model the user's intention to use a recommended command. In the future we will evaluate the applicability of UTAUT to IDE command acceptance, but we presume that performance expectancy, effort expectancy, and social influence are all determining the user acceptance of a recommendation. The user's perception of these three properties is likely influenced by various contextual factors. For example, a command will be perceived as more useful and easier to use if it can be applied instantly; its recommendation is also more persuasive if other people are using the command in a similar context. We identified four types of contextual factors that will likely affect the intention to use the command: current practice, environment, interaction, and recommendation presentation context (in the following subsections these context types are discussed more in detail). We suggest the following model:

$$User \times Command \times Context \rightarrow R_i \times R_p \times R_e \times R_s \quad (2)$$

where R_i stands for intention to use (i.e., overall) rating, and R_p, R_e, and R_s represent performance expectancy, inverse effort expectancy, and social influence (i.e., individual criteria) ratings, respectively. To ease the calculations, we assume that ratings are normalized to $[0, 1]$ interval, where higher values indicate better performance expectancy, lower effort expectancy, positive social influence, and higher probability that the user will use the command.

We assume that quantity of manual effort required to use a RS and the number of developers who will use it are inversely related. It means that it will be very hard to collect any kind of explicit rating, and especially ratings for each criterion. It is probably better to regard performance expectancy, inverse effort expectancy, and social influence as hidden determinants, and only collect binary ratings of intention to use, i.e., if the command was used the value is 1, otherwise it is 0; in that way, as it is common in RSs, ratings can be collected not only after recommendation presentations, but also we can record *positive ratings* whenever the command is used. Nevertheless, we will have to conduct some experiments and studies to find out whether this kind of data is sufficient for highly accurate recommendations. This approach also limits the number of potentially applicable algorithms that can be used for recommendation generation.

We believe that context should be monitored automatically as well. In figure 2 a draft of the expected RS components is presented. *Context Detector* and *User Profile Detector* monitor the $User$ and $Context$ factors and provide data to the *Recommender* component, which contains knowledge about commands and performs calculations that generate command recommendations. The *Recommender* and *Presentation* component communicate in two-ways, because recommendation generation is affected by the presentation and generated recommendations also affect the presentation. *Presentation* component is responsible for adapting the visualization and collecting ratings, which is in our case either a command recommendation rejection or a command use. Presentation is adapted to a user profile as well.

A. Current Practice Context

If we reconsider Fischer's partition of functionality types (see figure 1), we can say that there are only three types of commands that have to be considered in the recommendation process. The first type represents commands that are well known, easily applied, and used regularly. The second type of commands is related to concepts that are vaguely known and used only occasionally, so it is possible that the user is aware of the command, but she is not using it for any reason (e.g.,

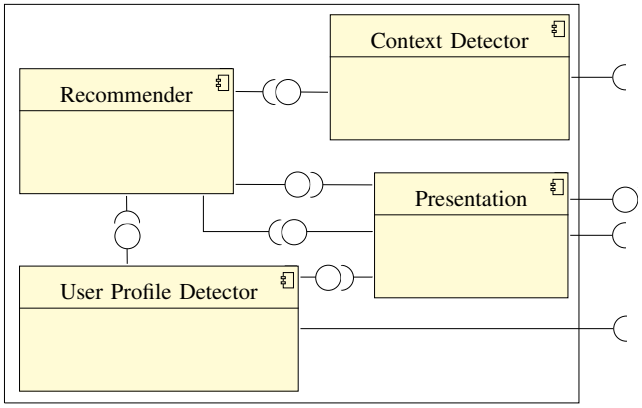

Fig. 2. UML component diagram of expected RS components.

it is too hard to remember or she does not find it useful) or she is executing it in a different, potentially suboptimal way (e.g., she uses a menu button instead of using a shortcut). The third type of commands are unknown ones and they represent the functionality of the system that the user has never encountered[4].

It is important to classify commands according to these three types, since recommendations of well known commands would only distract the user, i.e., first type of commands should not be recommended. When the system recommends a vaguely known command, the user will evaluate it on the base of her current practice, i.e., if a command of the second type is recommended, values of R_p, R_e, R_s, and R_i are not *absolute*, but *relative* with respect to current practice. On the other hand, when an unknown command is recommended, the user will accept the command if (still subjectively assessed) values of R_p, R_e, R_s, and R_i are acceptably high, independently of a current practice context. For example, if *generate getters and setters* shortcut is recommended and the developer did not know that she can automatically generate this type of methods, she will find it more valuable than if she always uses menu button, which takes more time than using a shortcut, but is easier to remember than a key sequence; even though, the objective value of the command is the same in both scenarios.

B. Environment Context

Opened windows, editors, files under development (their type and content), unresolved errors, quick fix recommendations, etc., define developer's environmental context; and we presume that an intention to use a command is affected by it. For example, it is more useful to use *generate getters and setters* command when there are more variables in an entity class.

C. Interaction Context

In addition to the environment context, we should take into account also the developer's interaction with the environment. The developer's navigation over source code files, recently executed commands, file modifications, current task (if it can be recognized), etc., influence intention to use a command similarly as the environment context factors. For example, the developer will perceive *generate getters and setters* command more useful if she has just created a set of variables, and less useful if she is trying to resolve a merging conflict.

[4]Unknown commands are $D4 \cap D2^C$, since at the moment when a recommended command is perceived by the user we cannot distinguish anymore between $D4 \cap D3$ and $D4 \cap D3^C$ (see figure 1).

D. Recommendation Presentation Context

According to [12], RS designer can aim at *choice support* or *persuasion* when visualizing recommendations. A RS is supporting choice when it presents options in a way that the user will be satisfied with the choice; on the other hand, a RS is persuading the user if it presents options in a way that the user will more likely select a specific option.

We believe that in command RS both goals should be pursued simultaneously: RS should recommend a command that will improve developer's performance and it should present it in a way that will increase the likelihood of selecting the command. It means that the user's choice patterns and the RS's choice support strategy are important context factors, since they both affect R_p, R_e, and R_s, and define the impact of each criterion on R_i, which is ultimately the most important one and easily observable.

We presume that a recommendation presentation that is congruent with the user's applied choice pattern (or combination of patterns) increases R_p, R_e, R_s, and R_i values, while improper presentation decreases them. For instance, consider a user that is applying the experience-based choice pattern and had bad experiences with the recommendations of colleagues and good experiences with the RS's recommendations. If the RS emphasizes that many colleagues are using the command, the recognition-primed effect could result in instant rejection, while emphasizing the fact that user liked previous RS's recommendations would increase the likeliness of recommendation acceptance. Table I contains examples of recommendation presentations depending on the ASPECT choice patterns and the command type.

IV. Conclusions and Future Work

In this article we presented a new multi-criteria context-aware rating prediction model for command RSs. The model is still in a preliminary stage; it is the result of discussions and findings of similar domains. The model effectiveness has to be evaluated by experiments that we are planning to conduct.

The model was inspired by the UTAUT model, which was designed for the assessment of the usage of a new information system. We did not find any study that addressed the applicability of UTAUT to estimate feature acceptance. We are only aware of [14], where the extended technology acceptance model (TAM) [15] – UTAUT predecessor – was used to study the acceptance of specific technology features by internal auditors; the results showed that perceived usefulness has more influence on feature acceptance when basic features are used, and perceived ease of use has more impact on feature acceptance when advanced features are used. We see

ASPECT choice pattern	Example	
	Vaguely known command (shortcut recommendation)	Unknown command
Attribute-Based	RS visualizes advantages: user does not have to lift the hands from the keyboard, does not have to search over menus, it is easier to remember this specific key sequence than to remember in which submenu the command button is.	RS visualizes performance and effort ratings.
Consequence-Based	RS visualizes how much time is saved every time the shortcut is used compared to current practice, and how much time will the user save per week if she always uses the shortcut.	RS describes what the command functionality is, i.e., what will happen if the command is selected.
Experience-Based	RS informs the user how many shortcuts she is already using and to which shortcuts the recommended one is similar.	RS visualizes how many and which commands were recommended in the past and which commands she liked.
Socially-Based	RS visualizes how many software developers recently started to use the shortcut and lists people who are familiar to developer.	RS visualizes how many times the command was recommended and how often it was executed.
Policy-Based	RS visualizes with which policies the use of the shortcut complies and with which the current practice does not.	RS visualizes with which policies the use of the command complies.
Trial and Error-Based	RS provides the functionality that can restore the previous state, i.e., the shortcut can be used in a "sandbox".	RS provides the functionality that can restore the previous state, i.e., the command can be used in a "sandbox".

TABLE I
EXAMPLES OF RECOMMENDATION PRESENTATION BASED ON ASPECT CHOICE PATTERN AND THE COMMAND TYPE.

no obvious argument against applying UTAUT to model the intention to use a specific functionality or a command. It is possible that the model is underfitting, but more research has to be done to generate enough evidence and potentially better models.

Our next step will be the analysis of IDE commands; we will focus on Eclipse, since it is an open source IDE and we know it better than other IDEs. We would like to create a credible list of *fine-grained* context factors, which will enable the implementation of a context monitoring RS component. We have not designed the component yet, but we expect that it will be composed of a set of Eclipse plug-ins that will automatically collect developers' interactions with the IDE (Eclipse UDC [16]), user's navigation and editing (Mylyn [17]), properties of artifacts under development, and other contextual data that will be identified as potentially relevant. We believe that data analysis and recommendation generation will have to be executed remotely, since client-side execution can slow down a development environment [18].

We will also analyze existing context-aware and multi-criteria rating prediction algorithms that can be applied to our model and we will further study learning styles and choice support strategies, so that we will be able to develop a RS prototype based on the latest research findings. We will develop a prototype to assess the command recommendation accuracy and effects on software development process in laboratory and industrial settings. Ultimately, we will evaluate the *overall* user experience of the RS. It is known that the accuracy of recommendations only partially describes the user experience, and that is the reason to use a user-centric framework for RS evaluation. We plan to use the framework described in [19].

REFERENCES

[1] G. Fischer, "User modeling in human-computer interaction," *User Modeling and User-Adapted Interaction*, pp. 65–86, 2001.

[2] E. Murphy-Hill, "Continuous social screencasting to facilitate software tool discovery," in *International Conference on Software Engineering*, 2012.

[3] F. Ricci, "Recommender systems: Models and techniques," in *Encyclopedia of Social Network Analysis and Mining*. Springer, 2014, pp. 1511–1522.

[4] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, *Recommendation Systems in Software Engineering*. Springer Berlin Heidelberg, 2014.

[5] F. Linton and H.-P. Schaefer, "Recommender systems for learning: Building user and expert models through long-term observation of application use," *User Modeling and User-Adapted Interaction*, pp. 181–208, 2000.

[6] J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice, "Communitycommands: Command recommendations for software applications," in *ACM Symposium on User Interface Software and Technology*, 2009.

[7] E. Murphy-Hill, R. Jiresal, and G. C. Murphy, "Improving software developers' fluency by recommending development environment commands," in *ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, 2012.

[8] P. Viriyakattiyaporn and G. C. Murphy, "Improving program navigation with an active help system," in *Conference of the Center for Advanced Studies on Collaborative Research*, 2010.

[9] G. Adomavicius, B. Mobasher, F. Ricci, and A. Tuzhilin, "Context-aware recommender systems," *AI Magazine*, pp. 67–80, 2011.

[10] G. Adomavicius, N. Manouselis, and Y. Kwon, "Multi-criteria recommender systems," in *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer, 2011, pp. 769–804.

[11] V. Venkatesh, M. G. Morris, G. B. Davis, and F. D. Davis, "User acceptance of information technology: Toward a unified view," *MIS Quarterly*, pp. 425–478, 2003.

[12] A. Jameson, "Recommender systems as part of a choice architecture for HCI," in *International Workshop on Decision Making and Recommender Systems*, 2014.

[13] N. Mandel and E. J. Johnson, "When Web pages influence choice: Effects of visual primes on experts and novices," *Journal of Consumer Research*, pp. 235–245, 2002.

[14] H.-J. Kim, M. Mannino, and R. J. Nieschwietz, "Information technology acceptance in the internal audit profession: Impact of technology features and complexity," *International Journal of Accounting Information Systems*, pp. 214–228, 2009.

[15] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quarterly*, pp. 319–340, 1989.

[16] https://eclipse.org/org/usagedata/.

[17] http://eclipse.org/mylyn/.

[18] A. Sillitti, A. Janes, G. Succi, and T. Vernazza, "Collecting, integrating and analyzing software metrics and personal software process data," in *Euromicro Conference*, 2003.

[19] B. P. Knijnenburg, M. C. Willemsen, Z. Gantner, H. Soncu, and C. Newell, "Explaining the user experience of recommender systems," *User Modeling and User-Adapted Interaction*, pp. 441–504, 2012.