GUI Design for IDE Command Recommendations

Marko Gasparic, Andrea Janes, Francesco Ricci, Marco Zanellati
Free University of Bozen-Bolzano
Dominikanerplatz 3, 39100 Bolzano, Italy

ABSTRACT

This paper describes a novel design of a graphical user interface (GUI) to recommend useful command within an integrated development environment. The recommendation GUI contains a description of the suggested command, an explanation why the command is recommended, and a command usage example. The proposed design is based on the analysis of relevant guidelines identified in the literature. Its perceived usability and acceptance were evaluated in a live user study with 36 software developers. Our findings, partially contradicting existing literature, indicate that the presentation of the command—the description and the example—is perceived as more useful than the explanation of the rationale for the recommendation.

Author Keywords

Integrated Development Environment; User Interface; Command; Recommender System

INTRODUCTION

High-functionality applications are complex systems that serve the needs of a diverse user population, providing a potentially overwhelmingly large set of functions [6]. While it is not expected that a generic user accesses all the provided functionality, for some applications the rate of the used functionality is surprisingly low [14, 19]. One of the reasons for such a behaviour is the lack of knowledge of which functionality is available and potentially useful [4, 9]. As a consequence, many users do not exploit the full potential offered by the application.

In order to improve the knowledge of the application functionality, recommender systems (RSs) have been proposed. RSs are personalized information search and filtering tools that direct the users to items that are estimated to be useful for them [23]. They are primarily conceived to support individuals who lack sufficient knowledge or time to evaluate a large set of available options [22, 24]. In this paper, we focus on command recommendations in an integrated development environment (IDE). We envision the following use case: while developers use an IDE, the command RS is observing their interaction with the application; if the recommender detects that a specific command would be useful, but is never used, it invites the developer at a convenient moment, e.g., when she has time, to learn and use this command in the future. To date, a set of algorithms for generating IDE command recommendations have been devised by Murphy-Hill et al. [20] and Zolaktaf and Murphy [32], but we are not aware of any well motivated and validated user interface to present these recommendations. Nevertheless, the recommendations must be presented in a graphical user interface (GUI) that allows the user to evaluate and act upon them [21]. We stress that the presentation itself can play an important role in the usage of a RS, such as: it can affect the user's trust and loyalty to the system, and it can influence the user's final decision to accept the recommendation [27]. We propose a GUI that is intended to present *global* suggestions [16], which are usually based on the entire command usage history and suggest commands that are relevant in general (as opposed to *opportunistic* suggestions [16], which are only relevant at a particular moment), but it is not bound to a particular recommendation algorithm. In this paper, we focus on how the recommendations should be presented once the attention has been acquired.

The contributions of this paper are: a) a new GUI design of an IDE command RS, which is based on design guidelines identified in the literature, and b) the evaluation of its perceived usability and acceptance, performed with 36 software developers. The proposed design consists of three parts: the command description and the usage example, both presenting the recommended item, and the explanation, providing a rationale for the recommendation. Our results show that most of the interviewed software developers would like to use an IDE command RS if it is augmented with a convenient GUI, such as the one proposed in this paper. Moreover, the results indicate that the acceptance of the GUI could be improved further, if it was adapted to the command and the recipient's profile. Finally, we discovered that study participants find the description of the command and its usage example more valuable than the explanation of the reasons for the recommendation. This finding disagrees with the outcome of previous research focused on traditional RSs (e.g., [10] and [25]); however, it is supported by the observations of Viriyakattiyaporn and Murphy [29]. We argue that this behavior occurs because almost half of the users do not perceive such explanations as useful.

RELATED WORK

The proposed GUI is influenced by the guidelines suggested by Murphy-Hill and Murphy [21], who have identified five

IUI 2017, March 13–16, 2017, Limassol, Cyprus.

http://dx.doi.org/10.1145/3025171.3025200

important success factors for an interface of a recommendation system for software engineering (RSSE). We directly addressed three of them: *understandability* (i.e., the user can identify *what* is recommended), *transparency* (i.e., the user can identify *why* it is recommended), and *assessability* (i.e., the user can assess if the recommendation is relevant).[1] We are not aware of any other application of the success factors presented in [21]. Furthermore, the GUI is influenced by the ASPECT and ARCADE models [12], which are reference models for understanding and supporting human choice in human-computer interaction, in the field of RSs. The ASPECT model defines six basic choice patterns that people apply when taking the decisions and the ARCADE model is a synthesis of high-level strategies that a RS can use to improve the quality of those decisions. We applied a set of the ARCADE strategies to directly support *attribute-*, *socially-*, *experience-*, and *consequence-based* choice patterns.[2] In addition to [21] and [12], we also followed the usability guidelines in a human-computer dialog, provided by Molich and Nielsen [18], and *argumentative facts* recommendation explanation approach, i.e., "A, B therefore C", suggested by Zanker and Schoberegger [31].

Existing IDE command RSs are Spyglass [29], which recommends navigation commands in IBM RTC [11], and Vignelli [26], which detects design flaws in the code and recommends suitable refactoring commands in IntelliJ IDEA [13]. But, since these RSSEs have been tailored to suggest specific types of commands, their GUI design cannot be directly applied to other types of IDE commands. On the other hand, various systems providing command recommendations within other high-functionality applications have been developed: OWL [15] for Microsoft Word [17], CommunityCommands [16] for AutoCAD [2], and two RSs developed by Wiebe et al. [30] for GIMP [8], namely, *social* and *task-based* RSs. These tools are not focusing on IDE command recommendations, but the criteria that an effective recommendation presentation must satisfy are somewhat similar. Observing these GUIs from the perspective of the three factors proposed in [21], we can say that they all support *understandability* and *assessability*, by providing short (OWL and CommunityCommands) and long command descriptions (GIMP RSs); while OWL and the *social* GIMP RS support also *transparency* in a form of the command popularity. The main distinctive features of our GUI are: it can be used to present IDE commands of any type, it provides explicit and personalized explanation why the command is recommended, and it provides a command usage example.

COMMAND RECOMMENDATION GUI

Fig. 1 shows the proposed GUI for recommending, for instance, the Open Resource command. The GUI consists of three parts, which are based on the guidelines summarized in Tab. 1. The presentation starts with the command name, followed by its shortcut, to enable more experienced users to execute it faster, which is in compliance with [18]. Next is a brief, single sentence description of the command, which supports *understandability* and *assessability* [21], by informing the user about the benefit and the functionality of the command. For instance, in Fig. 1, the Open Resource command "enables quick finding and opening of project resources". The description is formulated according to the guidelines of Molich and Nielsen [18], i.e., avoiding rarely needed information, expressing the message clearly, using familiar words, and minimizing the user's memory load.

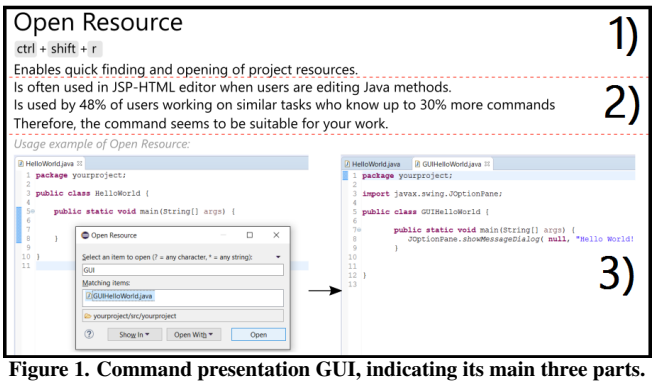

Figure 1. Command presentation GUI, indicating its main three parts.

Guideline	Part 1	2	3
Support understandability [21]	×		×
Enable transparency [21]		×	
Support assessability [21]	×	×	×
Support a attribute-based choice pattern [12]		×	
Support a socially-based choice pattern [12]		×	
Supporta an experience-based choice pattern [12]		×	
Support a consequence-based choice pattern [12]			×
Access information and experience [12]		×	×
Represent the choice situation [12]		×	×
Combine and compute [12]		×	
Evaluate on behalf of the chooser [12]		×	
Use simple and natural language [18]	×		
Minimize user's memory load [18]	×		
Provide shortcuts [18]	×		

Table 1. Guidelines for each GUI part and the supporting literature.

To enforce *transparency* [21], part 2 contains an explanation of why the command is recommended. It starts with a description of the context in which the command is often used, by illustrating in which GUI element, during which activity, on which artifact, and after which other activity the command is typically applied. For instance, in Fig. 1, the Open Resource command "is often used in JSP-HTML editor when users are editing Java methods". This context, which is identified by the RS, also represents the situations in which the recommendation recipient often works. In that way, the context description supports *assessability* [21] and the *experience-based* choice pattern [12]. By making the situation recognizably similar to the previous experiences of the user, the GUI also complies with the *represent the choice situation* strategy [12]. We note that the exact identification of the relevant

[1]The *distraction* and the *trust* [21] factors are out of scope of this study: the first because we are focusing on the user interaction with the system only after the attention was already acquired and the second because it is typically affected by many other aspects of a RS.

[2]We do not address the remaining choice patterns because the support for the *trial-and-error* is already provided in an IDE by the Undo command and the *policy-based* pattern does not apply.

context factors may be delegated to a specific context model, as that proposed by Gasparic et al. [7]. For the purposes of this study, we manually identified the relevant context.

The second sentence in part 2 indicates the proportion of users who already use this command and are similar to the target user, or slightly more proficient. For instance, in Fig. 1, the Open Resource command "is used by 48% of users working on similar tasks who know up to 30% more commands". This sentence follows the *access information and experience* and the *combine and compute* ARCADE strategies [12]. It also supports *socially-* and *attribute-based* choice patterns [12], by providing the information that emphasizes the sense of identification and affiliation of the user to a group of people and by numerically expressing the level of command popularity. In this study, we arbitrarily defined the proportions.

The third sentence in part 2: "Therefore, the command seems to be suitable for your work.", is the conclusion of the argument, structured according to the *argumentative facts* explanation approach [31]. While the first part of the argument leverages the command popularity and implies that the context in which the command can be used is also a common usage context of the user, the last sentence, which is always the same, adopts the *evaluate on behalf of the chooser* strategy [12], by implicitly urging the user to use the command.

In part 3, which is—as part 1—dedicated to the command presentation, we placed a graphical usage example consisting of two screenshots: the first shows a typical situation in which the proposed command can be used and the second shows how executing the command affects that situation. The example follows the *access information and experience* and the *represent the choice situation* ARCADE strategies [12], by providing the information about the initial state in the first image and "a preview of the experience of executing a particular action" [12] in the second image. In that way it also supports *understandability* and *assessability* [21] and the *consequence-based* choice pattern [12]. For the purposes of this study, we manually generated usage examples.

EVALUATION

We conducted a survey among the participants of XP 2016: a scientific conference with a strong participation of practitioners from the industry. Our goal was to test whether the proposed GUI would be well accepted by the target population, to understand the importance of the different GUI parts, and to identify potential GUI flaws and opportunities for improvements. In the survey we gathered demographic data, the general attitude towards IDE command recommendations, and the familiarity with nine editing, navigating, and debugging sample commands from Eclipse [5] (presented in a random order), and we assessed how the recommendation of the *least familiar command* was perceived.

The format of the survey was "paper and pencil". The questionnaire started with 3 non-compulsory fields: email address, gender, and age; all the remaining fields were compulsory. We asked participants to state their programming experience (in years) and to list the activities they perform in an IDE. We asked participants to list the IDEs and the programming languages they used in the last two years. After that, they had to mark, on a five-point Likert scale, how strongly they agree with the following statements:

- "It would be useful for me to learn more commands available in my IDE";
- "It would be useful for most of the IDE users to learn more commands"; and
- "I think that using an IDE command recommender system would be a good idea".

In the second part of the questionnaire, for each of the nine sample commands, we asked the participants if they ever used the command before (in any IDE) and how familiar they are with it, for which we provided four options (following the taxonomy proposed by Anderson-Meger [1]):

- "I have never heard of it";
- "I have some idea what it is";
- "I have a clear idea what it is"; and
- "I can explain what it is".

In the third part, we presented a fictional scenario of a situation in which the command recommendation appears on the bottom-left side of the screen when the user starts up the IDE, and if the user selects the recommendation, the here proposed command recommendation GUI becomes visible in the center of the screen. For the study purposes, we printed the mock-ups of the application interface and attached them to the questionnaire. Each participant had to select a command from the list of commands presented before and imagine that this is the recommended command. We asked them to select a command they do not know yet or a command that is the most interesting to them, if they already know all the commands. Then they had to express their agreement with three statements from the System Usability Scale (SUS) [3] and with three statements from the Unified Theory of Acceptance and Use of Technology model (UTAUT) [28], using a five-point Likert scale. The statements are listed in Tab. 2, together with the responses of the participants. Finally, we asked survey participants which parts of the GUI—"command name and description", "explanation of recommendation", and "usage example" (see Fig. 1)—they find necessary and which they find useful for evaluating the quality of the command recommendation, and if they think that some information is missing. A sample of the questionnaire is publicly available at: `https://sites.google.com/site/mgasparic/gui`.

Results and Discussion

We distributed approximately 100 questionnaires and we received back 40, out of which 36 had all the compulsorily questions answered and were therefore included in the analysis. Five participants are female, and the age range is between 18 and 64. Four participants have less than 2 years of programming experience, 3 have more than 30 years, and others have between 2 and 30. All except one participant—who is not using any IDE—are editing and navigating in an IDE, 32 participants are also debugging, and 20 are using version control. 20 participants used Eclipse recently, which means that approximately one half of the participants should be familiar with the environment in which the GUI mock-ups were presented. 28 participants were recently programming in Java,

thus, more than three quarters of the participants know the programming language in which all the selected commands are applicable and is used also in the command usage examples. 30 participants agree or strongly agree with the statement that it would be useful for them to learn new commands; 34 think that it would be useful for others. 29 agree or strongly agree that an IDE command RS is a good idea, 6 are neutral, and only 1 disagrees.

In general, the participants have a good knowledge of IDE commands: 12 participants used in the past all nine commands included in the questionnaire and only two used less than 5 of those commands; additionally, the most common familiarity levels are "I have a clear idea what it is" and "I can explain what it is". As we report in Tab. 2, most participants perceive the recommended command as easy to use and the majority also think that other people would quickly learn how to use this command, however, there is a disagreement about the presentation: 36% (8%+28%) think that they would still need some additional support to learn how to use the command and 39% (31%+8%) think they would not. A minority, 14% (8%+6%) of the participants, think that our GUI is unnecessarily complex. On the other hand, 31% think that there is still some information missing in the presentation; they suggested to add the following information: a list of commands similar to the recommended one or commands that are used with it, more usage examples, step-by-step instructions, a video describing the command, and a "stop recommending this command" option. In replying to the question about which parts of the GUI were considered necessary and/or useful, the command description and the usage example were perceived as useful by almost 90%, but only 53% perceived the explanation as useful. To further stress the importance of the description and the example, in comparison with the recommendation explanation, we note that the command description is perceived necessary by 83% and the usage example by 75%, but the explanation is perceived necessary only by 36% of participants. Nevertheless, the acceptance of recommendations was high: three quarters of participants would find the command useful in their job and almost two thirds plan to use it in the future.

Statement	SA	A	N	D	SD
	(Answers in %)				
I would find the command easy to use.	19	67	8	3	3
I think I would need some additional support to actually learn how to use the command.	8	28	25	31	8
I would imagine that most people would learn to use this command very quickly.	14	44	22	14	6
I find the command presentation unnecessarily complex.	6	8	36	47	3
I would find the command useful in my job.	19	58	14	6	3
I intend to use the command in the future.	25	39	28	6	3

Table 2. Results of the agreement with UTAUT and SUS statements.

By studying the survey results we observed also that: 1) there is a strong tendency among those that think that others should learn more commands to consider a command RS as a good idea; 2) those participants who think that they should learn more commands also perceive the recommendations as more useful, compared to the others; 3) those participants who find the command easy to use also tend to find the presentation less complex; 4) older participants are more likely to perceive the presentation as unnecessarily complex than younger, regardless of their programming experience; and 5) the need for an additional support varies considerably between the commands. The observations 1 and 2, and the high proportion of the interviewees who would like to learn more commands indicate that such a system, if provided in practice, would be well accepted. The observations 3, 4, and 5 indicate that the ease of use and the need for an additional support depend on the command and on the user, thus, we conjecture that a certain level of the GUI customization could be beneficial.

Based on these results, we made a number of changes to the proposed GUI: 1) we now provide a link to a video illustrating the command usage, to offer clearer and more detailed instructions; 2) we place the recommendation explanation at the end, since it was perceived as less useful, compared to the other parts; and 3) we modify the explanation text, since it was perceived complex, not so useful, and—according to the side note on one questionnaire—even offensive. The updated GUI is presented in Fig. 2.

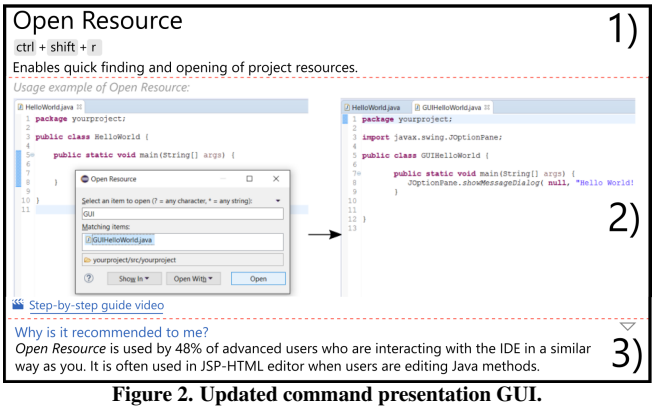

Figure 2. Updated command presentation GUI.

CONCLUSIONS AND FUTURE WORK

This paper describes the first GUI explicitly designed for recommending different types of IDE commands. Moreover, it is the first GUI for recommending commands in high-functionality applications that contains personalized explanation or usage example. The design was motivated by a set of guidelines that we identified in the literature. Furthermore, we analyzed the acceptance and the perceived usability of the GUI in a live study with 36 software developers. The results show that the GUI is well accepted by the study participants and that the command presentation is perceived more useful than the explanation of the recommendation rationale. We note that in this analysis the recommendations were not generated and personalized by a recommendation algorithm and that we did not measure the user response to the recommendation while using an IDE. In the future, we plan to evaluate the proposed improved version of our GUI, and to address the above mentioned limitations.

REFERENCES

1. Anderson-Meger, J. *Why Do I Need Research and Theory?: A Guide for Social Workers*. Taylor & Francis, 2016.
2. Autodesk. AutoCAD. `http://www.autodesk.com/products/autocad/overview`, accessed October 10th, 2016.
3. Brooke, J. SUS: A "quick and dirty" usability scale. In *Usability Evaluation in Industry*. Taylor and Francis, 1996.
4. Campbell, D., and Miller, M. Designing refactoring tools for developers. In *Workshop on Refactoring Tools* (2008).
5. Eclipse contributors. Eclipse. `http://www.eclipse.org`, accessed October 10th, 2016.
6. Fischer, G. User modeling in human-computer interaction. *User Modeling and User-Adapted Interaction* (2001).
7. Gasparic, M., Murphy, G. C., and Ricci, F. A context model for IDE-based recommendation systems. *Journal of Systems and Software* (2016).
8. GIMP contributors. GNU Image Manipulation Program. `https://www.gimp.org/`, accessed October 10th, 2016.
9. Grossman, T., Fitzmaurice, G., and Attar, R. A survey of software learnability: Metrics, methodologies and guidelines. In *SIGCHI Conference on Human Factors in Computing Systems* (2009).
10. Herlocker, J. L., Konstan, J. A., and Riedl, J. Explaining collaborative filtering recommendations. In *ACM Conference on Computer Supported Cooperative Work* (2000).
11. IBM. Rational Team Concert. `http://www.ibm.com/software/products/en/rtc`, accessed October 10th, 2016.
12. Jameson, A., Berendt, B., Gabrielli, S., Cena, F., Gena, C., Vernero, F., and Reinecke, K. Choice architecture for human-computer interaction. *Foundations and Trends in Human-Computer Interaction 7* (2014), 1–235.
13. Jet Brains. IntelliJ IDEA. `https://www.jetbrains.com/idea`, accessed October 10th, 2016.
14. Lafreniere, B., Bunt, A., Whissell, J. S., Clarke, C. L. A., and Terry, M. Characterizing large-scale use of a direct manipulation application in the wild. In *Graphics Interface* (2010).
15. Linton, F., and Schaefer, H.-P. Recommender systems for learning: Building user and expert models through long-term observation of application use. *User Modeling and User-Adapted Interaction 10* (2000), 181–208.
16. Matejka, J., Li, W., Grossman, T., and Fitzmaurice, G. CommunityCommands: Command recommendations for software applications. In *ACM Symposium on User Interface Software and Technology* (2009).
17. Microsoft. Microsoft Word. `https://products.office.com/en-us/word/`, accessed October 10th, 2016.
18. Molich, R., and Nielsen, J. Improving a human-computer dialogue. *Communications of the ACM 33* (1990), 338–348.
19. Murphy-Hill, E. Continuous social screencasting to facilitate software tool discovery. In *International Conference on Software Engineering* (2012).
20. Murphy-Hill, E., Jiresal, R., and Murphy, G. C. Improving software developers' fluency by recommending development environment commands. In *ACM SIGSOFT International Symposium on the Foundations of Software Engineering* (2012).
21. Murphy-Hill, E., and Murphy, G. C. Recommendation delivery. In *Recommendation Systems in Software Engineering*. Springer, 2014.
22. Resnick, P., and Varian, H. R. Recommender systems. *Communications of the ACM 40* (1997).
23. Ricci, F. Recommender systems: Models and techniques. In *Encyclopedia of Social Network Analysis and Mining*. Springer, 2014.
24. Ricci, F., Rokach, L., and Shapira, B. Recommender systems: Introduction and challenges. In *Recommender Systems Handbook*. Springer, 2015.
25. Sinha, R., and Swearingen, K. The role of transparency in recommender systems. In *Extended Abstracts on Human Factors in Computing Systems* (2002).
26. Stuckemann, S. Vignelli - automated design guidance for developers. *Imperial College London, Department of Computing* (2015).
27. Tintarev, N., and Masthoff, J. Explaining recommendations: Design and evaluation. In *Recommender Systems Handbook*. Springer, 2015.
28. Venkatesh, V., Morris, M. G., Davis, G. B., and Davis, F. D. User acceptance of information technology: Toward a unified view. *MIS Quarterly* (2003), 425–478.
29. Viriyakattiyaporn, P., and Murphy, G. Challenges in the user interface design of an IDE tool recommender. In *Workshop on Cooperative and Human Aspects on Software Engineering* (2009).
30. Wiebe, M., Geiskkovitch, D. Y., and Bunt, A. Exploring user attitudes towards different approaches to command recommendation in feature-rich software. In *International Conference on Intelligent User Interfaces* (2016).
31. Zanker, M., and Schoberegger, M. An empirical study on the persuasiveness of fact-based explanations for recommender systems. In *IntRS Workshop* (2014).
32. Zolaktaf, S., and Murphy, G. C. What to learn next: Recommending commands in a feature-rich environment. In *International Conference on Machine Learning and Applications* (2015).