Information and Software Technology 92 (2017) 236–255

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A graphical user interface for presenting integrated development environment command recommendations: Design, evaluation, and implementation

Marko Gasparic[a,*], Andrea Janes[a], Francesco Ricci[a], Gail C. Murphy[b], Tural Gurbanov[a]

[a] Free University of Bozen-Bolzano, Dominikanerplatz 3, 39100 Bolzano, Italy
[b] University of British Columbia, 201-2366 Main Mall, Vancouver, BC V6T 1Z4, Canada

ARTICLE INFO

Article history:
Received 12 December 2016
Revised 30 July 2017
Accepted 16 August 2017
Available online 19 August 2017

Keywords:
Integrated development environment
User interface
Command
Functionality
Recommender system
Software development

ABSTRACT

Context: A set of algorithms exist to generate integrated development environment (IDE) command recommendations. The recommendations are aimed at improving software developer's interaction with an IDE. Even though the interface is a critical element of every recommender system, we are not aware of any existing graphical user interface to present such recommendations.
Objective: This paper describes and evaluates a novel design of a graphical user interface to recommend commands within an IDE. The interface contains a description of the suggested command, an explanation of why the command is recommended, and a command usage example.
Method: The proposed design is based on the analysis of guidelines identified in the literature. Its acceptance and usability were evaluated through a user study with 36 software developers and semi-structured interviews with 11 software developers.
Results: The results indicate that the suggested interface is well accepted, but it can be further improved. Through the interviews and the implementation of the interface, we identified a series of requirements important for the development of future IDE command recommender systems.
Conclusions: This paper shows that a convenient graphical user interface is critical to achieve high acceptance of IDE command recommendations. Our work also illustrates steps useful for undertaking user studies related to IDE command recommendations in a practical setting without human intervention. A future step is to evaluate the interface within the business environment, where recommendations are generated and presented in an IDE used by practicing software developers as part of their normal workday.

1. Introduction

To serve the needs of a diverse user population, high-functionality applications provide a large set of functions that are potentially overwhelming for a user [1]. Although it is not expected that a generic user access all the provided functionality, for some applications—including integrated development environments (IDEs)—the proportion of the used functionality is surprisingly low. For example, average users of Eclipse IDE[1] use only 42 out of more than 1100 available commands [2]; where a command corresponds to a menu button or a shortcut that executes a function, such as paste or open resource, amongst many others.

One of the likely reasons for the low usage of functionality is the lack of the exact knowledge of which functionality is available and potentially useful [3,4]. As a consequence, many users do not exploit the full potential offered by an application. Conversely, a better knowledge of the target high-functionality application can help users to more effectively choose the precise functionality that is more useful in a specific situation. To improve such knowledge, the application of recommender systems (RSs) has been proposed. Recommender systems are personalized information search and filtering tools that direct the users to items that are estimated to be useful for them [5]. They are primarily conceived to support individuals who lack sufficient knowledge or time to evaluate the potentially overwhelming number of alternative options that may be available [6,7].

* Corresponding author.
E-mail addresses: marko.gasparic@stud-inf.unibz.it, m.gasparic@gmail.com (M. Gasparic).

[1] http://www.eclipse.org.

http://dx.doi.org/10.1016/j.infsof.2017.08.006
0950-5849/© 2017 Elsevier B.V. All rights reserved.

In this paper, we focus on command recommendations in an IDE. Within software engineering, it is generally accepted that software development tools can affect the efficiency and quality of software construction [8–10] and that the knowledge of those tools directly impacts on the productivity of programmers [11]. One reason for this is that "tools allow repetitive, well-defined actions to be automated, reducing the cognitive load on the software engineer who is then free to concentrate on the creative aspects of the process" [12, pp. 368]. Thus, we conjecture that a better knowledge of the commands offered by an IDE would help programmers to more effectively choose the functionality that can help them in a specific situation.

We envision the following use case: while developers use an IDE, the command recommender system is observing their interaction with the application; if the recommender detects that a specific command would be useful, but it is never used, it invites the developer, at a convenient time and with a well suited graphical user interface (GUI), to learn and use this command in the future. In this scenario, it is important to design an appropriate GUI to present the recommended commands in a way that allows the user to evaluate and act upon them [13].

To date, a set of algorithms for selecting IDE command recommendations has been proposed by Murphy-Hill et al. [14], Zolaktaf and Murphy [15], and Gasparic et al. [16]; but we are not aware of any specifically designed and validated user interface to present these recommendations. Nevertheless, the user interface used to present recommendations is a critical element of a recommender system, 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 [17].

We propose and validate a GUI designed for an IDE command recommender system, which is based on design guidelines identified in the literature. The proposed GUI consist of three parts: command name and description, explanation of recommendation, and command usage example. The GUI is not bound to a specific type of IDE commands nor to a particular recommendation algorithm, hence, it is general and can be supported by a variety of command recommender systems. The specific research goals addressed in this paper are:

- design a novel GUI to recommend IDE commands based on approaches and guidelines identified in the literature; and
- evaluate the acceptance and usability of the proposed GUI.

To design the GUI, we followed Design Science research guidelines [18]. We evaluated the perceived usability and acceptance of our GUI by performing a user study with 36 software developers and semi-structured interviews with 11 software developers. The results show that most of the participants would like to use an IDE command recommender system if it is augmented with a convenient GUI, such as the one proposed in this paper. The results also indicate that the acceptance of the GUI can be improved further by adapting the GUI to the recommended command and the recipient's profile. In particular, the developers would like to be in control of the information that is included in the presentation of the command, some would like to block certain types of recommendations, and some would like to be able to customize the types of data that are shared with the recommender system. Finally, we discovered that users find the description of the suggested command and its usage example more valuable than the explanation of the reasons for the recommendation.

Based on the evaluation results, we updated the GUI and implemented a prototype that can automatically generate personalized recommendation explanations. Additionally, we describe in this paper a potential approach to automate the generation of other parts of the GUI content, such as command descriptions and usage examples. The full automation of the generation of the content for presenting IDE commands will enable the set of recommendable commands to grow together with the IDE.

The main contributions of our work are: the development of a new GUI design for an IDE command recommender system based on guidelines identified in the literature, its evaluation in a form of a user study, and a road-map towards a fully automated IDE command recommendation presentation. A brief explanation of the GUI and initial survey results are also reported in a short paper published at the International Conference on Intelligent User Interfaces [19].

The reminder of the paper is structured as follows: the research method is discussed in detail in Section 2; the proposed GUI is presented in Section 3; the guidelines identified in the literature to design GUIs of recommender systems are presented in Section 4.1; the list of existing command recommender systems and their assessment in respect to the main guidelines are in Section 4.2; the results of the evaluation are in Section 5; the discussion of the results is in Section 6; the future work required for putting the GUI into practice, including a road-map towards a fully automated generation of the GUI content, is presented in Section 7; and the conclusions are in Section 8.

2. Research method

In this section, we describe the Design Science paradigm and explain how we followed the guidelines devised by Hevner et al. [18].

In general, engineers in various fields are devising artifacts that have desired properties to attain specific goals [20]. Unlike traditional natural sciences, which are focused on understanding the reality, the focus of Design Science is the creation of artifacts that serve human purpose and are assessed according to their value or utility [21]. Fundamentally, Design Science is a problem-solving paradigm, which combines existing theories with experience, creativity, and intuition of the researcher, to solve problems arising from the organization of people and technology [18].

Design Science consists of two basic activities: building and evaluating; hence, it is particularly applicable to computer science research, which is mainly concerned with artificial phenomena that can be both created and studied [21]. Moreover, due to the complexity of the artifact design and often insufficient existing theories, a Design Science approach is proactive, in the sense that a new artifact is first created with theories focused on its application impact following after [18]. According to the literature review and typology of Offermann et al. [22], who studied 62 research papers published in MIS Quarterly journal and in the proceedings of the international conference on Global Perspectives on Design Science Research, there are eight basic types of artifacts that emerged during the application of Design Science paradigm, namely: *system design, method, language/notation, algorithm, guideline, requirements, pattern*, and *metric*. The overall goal of this paper is to design and evaluate a GUI to recommend useful commands within an IDE, thus, the type of the artifact described in this paper is *system design*.

Hevner et al. [18] devised guidelines to conduct Design Science, which state that Design Science research requires the creation of an innovative, purposeful artifact (guideline 1) for a specified problem domain (guideline 2). To understand if the artifact helps to solve the specified problem, a thorough evaluation of the artifact is crucial (guideline 3). Moreover, the artifact must solve a so far unsolved problem or solve a known problem in a more effective or efficient way (guideline 4); the novelty requirement is what differentiates Design Science from design. The artifact itself must be rigorously defined, formally represented, coherent, and internally consistent (guideline 5). The process by which the artifact is created defines the problem space and describes the mechanism by which an effective solution is found (guideline 6). Finally, the results of the Design Science research must be communicated effectively (guideline 7) to a technical audience (researchers who will extend them and practitioners who will implement them) and to a managerial audience (researchers who will study them in context and practitioners who will decide if they should be implemented within their organizations).

The work of this paper is based on the fulfillment of these seven guidelines. We present the designed artifact (guideline 1) and the rationale we considered in its design (Sections 3 and 4.1). We describe the problem relevance (guideline 2), by presenting the potential usefulness of IDE command recommender systems, the importance of the GUI in such systems, and the lack of an existing GUI for IDE command recommender systems (Section 1). We evaluate the artifact (guideline 3) focusing on two aspects: the acceptance and the usability of the GUI (Sections 5 and 6). Additionally, we describe the acquired knowledge about the importance of different GUI parts, together with identified potential GUI flaws and opportunities for improvements. We adopt two instruments to perform the evaluation:

- a structured questionnaire to assess the acceptance and perceived usability of the designed GUI; and
- a semi-structured interview to gather the general attitude towards IDE command recommender systems, how such systems should work so that users would accept them, which information such systems should provide to the user, how a recommendation should be explained, and how such systems should notify a user when a new recommendation is available.

Between 24th and 27th May 2016, the first author of this paper distributed approximately 100 questionnaires and conducted 11 interviews among the participants of the XP 2016 conference (see [23] for the initial definition of the study). XP is a scientific conference with a strong participation of practitioners from the industry.

Using the taxonomy of design evaluation methods proposed in [18], we can categorize the questionnaire as "experimental" and "simulation", since we confronted the study participants with mock-ups of the designed GUI and collected their feedback on the GUI acceptance and the perceived usability. The semi-structured interviews are complementary to the questionnaires, since they help to understand the context in which the design artifact is evaluated.

A widely accepted approach for formulating measurement goals and systematically refining them into measures that specify the data to be collected is the Goal-Question-Metric (GQM) model [24,25]. We use it to define the data to be collected by the questionnaire and the interviews: we begin with structured measurement goal definition, continue with questions to describe how to achieve the measurement goal, and end with description of how to collect the data to answer the defined measurement questions, i.e., metrics.

The measurement goal of the questionnaire used for the GUI simulation study—formulated as a GQM-goal—is:

Analyze the proposed GUI
to evaluate it
with respect to its usability and acceptance
from the point of view of a software developer receiving an IDE command recommendation
in the context of using an IDE.

We defined four measurement questions associated with this goal, to specify the information we want to gain from the questionnaire:

Q1: Which types of developers—according to gender, age, knowledge, experience, and general attitude towards IDE command recommender system—participated in the evaluation of the proposed GUI?
Q2: How well do participants accept IDE command recommendations, if they are presented by the proposed GUI?
Q3: How well is the proposed GUI accepted by the participants?
Q4: How useful do the participants perceive the parts of the designed GUI?

To answer these measurement questions, we formulated corresponding questions within the questionnaire. The format of the questionnaire was "paper and pencil". It consisted of three groups of questions: *demographics* (Q1), *command knowledge* (Q1), and *perception of the proposed GUI* (Q2, Q3, and Q4). Before the XP conference, we performed a pilot study within the Faculty of Computer Science at the Free University of Bozen-Bolzano. We asked five researchers, who occasionally use an IDE, to answer the questionnaire. For these pilot participants, we measured the time needed to ensure that the questionnaire was not too long. We also collected their opinions to ensure that the questionnaire was understandable for average IDE users.

The questionnaire started with fields for email address, gender, and age (these were the only non-compulsory fields). After that, we asked participants to state their programming experience and typical activities they perform in an IDE. Then we asked them to list the IDEs and the programming languages they have used in the last two years. At the end of the first part, the study participants had to mark, on a five-point Likert scale, how strongly they agree with the statements about the usefulness of learning new IDE commands for themselves and for others, and whether they think that using an IDE command recommender system would be a good idea.

In the second part of the questionnaire, we presented nine sample commands and for each command we asked the participants if they have ever used the command before (in any IDE) and how familiar they are with it.[2] These questions allowed us to estimate the general IDE knowledge of the participants and forced the participants to carefully examine all nine commands, before selecting one in the third part of the questionnaire. The selection of sample commands was based on the data set of Gasparic et al. [27]. We selected three navigating, three editing, and three debugging commands. Each group included one command that was used very often, one that was used sometimes, and one that was used rarely by the subjects observed in the study described in [27]. These commands are:

- Breakpoint Properties (debugging, 3 executions).
- Navigate Back (navigating, 512 executions).
- Next Editor (navigating, 5 executions).
- Open Resource (navigating, 142 executions).
- Organize Imports (editing, 1810 executions).
- Rename Element (editing, 256 executions).
- Step Over (debugging, 6747 executions).
- Step Return (debugging, 114 executions).
- Surround with Try-Catch (editing, 5 executions).

We note that in each questionnaire the order of the commands was random, because we wanted to reduce command order bias.

In the third part of the questionnaire, which is the core of our study, we presented a fictional scenario of a situation in which the command recommendation list appears on the bottom-left side of the screen when an IDE opens. The commands in the list are presented only with the names, but after the user selects a recommendation, the corresponding command presentation becomes

[2] To express the familiarity with the command, the participants had to select one of the four options from the taxonomy proposed by Anderson-Meger [26]. For the exact list of options, please see Table A3, which appears in Appendix.

visible in the center of the screen. The command presentation is visualized within the GUI described in Section 3.

The participants had to imagine that one of the nine commands presented before was recommended to them. They had to select a command of which they were unaware or a command that is the most interesting to them, if they already know all nine. For the study purposes, we printed the mock-ups of the application interface and attached them to the questionnaire. We note that the usage of paper-based prototypes is not uncommon in human–computer interaction research [28,29].

After the participants selected a command, they had to answer the last set of questions. First, they had to express their agreement with three statements from the System Usability Scale (SUS) [30] and with three statements from the Unified Theory of Acceptance and Use of Technology (UTAUT) [31], which are related to the GUI and the acceptance of the recommendations. These six statements were ordered as follows:

- I would find the command easy to use (UTAUT).
- I think I would need some additional support to actually learn how to use the command (SUS).
- I would imagine that most people would learn to use this command very quickly (SUS).
- I find the command presentation unnecessarily complex (SUS).
- I would find the command useful in my job (UTAUT).
- I intend to use the command in the future (UTAUT).

Then we asked the participants which parts of the GUI—“command name and description”, “explanation of recommendation”, and “usage example” (see Fig. 2, Section 3)—they find necessary and which they find useful for evaluating the quality of the command recommendation. At the end of the questionnaire, we asked participants whether any information was missing.

The exact list of questions and the mapping to the GQM defined above are presented in Table A3, which appears in Appendix A sample of the questionnaire is publicly available at: https://sites.google.com/site/mgasparic/gui. The answers to the questions, which are the GQM-metrics associated with the GQM-questions, are presented in Section 5.1.

The questionnaires were available for the XP 2016 participants at the reception desk and on the chairs in the rooms in which programming workshops took place. Moreover, the questionnaires were distributed personally by the first author of this paper during the breaks between the sessions. The participants could return the completed questionnaire to the first author or throw it in a box available at the reception desk.

While the questionnaire was designed to specifically evaluate the designed GUI, by simulating its usage, we also wanted to interview software developers to understand their expectations about the recommendation delivery and the data used to generate a recommendation. The measurement goal of the interviews—formulated as a GQM-goal—is:

Analyze software developers (participating at the XP 2016 conference)
to characterize them
with respect to their expectations of recommendation delivery and the data used to generate recommendations
from the point of view of a software developer receiving an IDE command recommendation
in the context of using an IDE.

We defined four measurement questions associated with this goal, to specify the information we want to gain from the interviews:

Q5: Which types of developers—according to their attitude towards recommender systems and their experience—participated in the interviews?
Q6: Based on which data should the recommendation algorithm work to provide useful recommendations?
Q7: Which information about a recommended command is expected?
Q8: How should a recommendation be notified to the user so that the user keeps using the system?

To answer these measurement questions, we formulated the questions for interviews. The interviews were conducted face-to-face, using open-ended questions as a guidance, but neither the exact wording nor the order of the questions was determined ahead of time, which “allowed the researcher to respond to the situation at hand, to the emerging worldview of the respondent, and to new ideas on the topic” [32, pp. 111]. The initial script with the essential open-ended questions was created by the first author of the paper. It was updated according to the comments of three other authors and another domain expert from the Free University of Bozen-Bolzano, who was not otherwise involved in this study.

The form of the interview was the *systematizing expert* interview [33,34], which is a specific form of applying semi-structured interviews. Using this technique, we managed to “collect context information complementing insights coming from applying other methods” [34, pp. 166]; in our case, the other method was the questionnaire. Because the interviewees are supposed to represent a group of experts for a certain field or activity [34], the participants had to have experience in software development and IDE usage. As the analysis shows, almost all the interviewees currently work in industry or have worked in the past. Since we tried to obtain a diverse sample of developers, we invited XP 2016 participants of different age and gender, who were available to do the interview during the conference breaks or sessions.

We started the interviews by asking the respondents about their development experience and if they also filled in the questionnaire. We then asked if they find it useful or interesting to learn new IDE commands and if they would find an IDE command recommender system useful. The answers to those questions are related to Q5.

Furthermore, imagining such a system, we asked on what data should recommendations be based. And we continued asking the respondents about sharing the data of command usage and their working context, while using an IDE, such as sharing the data about opened views, source code meta-data, or even pure source code, for example. The answers to those questions were used as the metrics of Q6.

We also asked the participants which information about a command they would like to receive when the command is recommended. The answers to those questions were used as the metrics of Q7.

Finally, we asked how recommendations should be delivered and what the interviewees thought about being interrupted in their work because of an upcoming recommendation. We associated those answers with Q8.

The interviews were recorded with a mobile phone and manually transcribed. To analyze the content of the answers, we used *open* and *axial* coding methods [35]. *Open* coding helps researchers to extract concepts and define categories in terms of their properties and dimensions, when analyzing the data. *Axial* coding helps to reassemble the data that were fractured during *open* coding, by linking the categories at the level of properties and dimensions. For instance, during *open* coding, we assigned codes to the answers of the respondents that reflect the essence of the answer, e.g., “avoid interruptions” code was assigned to “I think interrupting can break the concentration and it might take a lot of time to get it back.” We note that at this step we did not deliberately reuse already assigned codes. Then, during *axial* coding, we grouped the extracted open codes into axial codes or *themes*; for example, we grouped

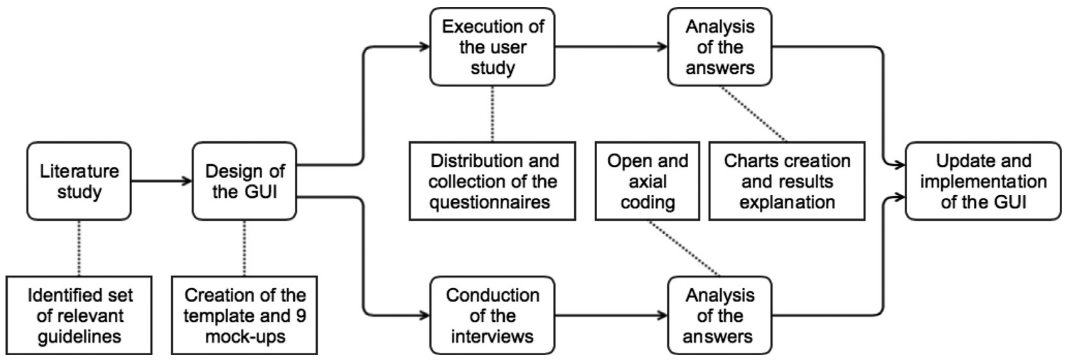

Literature study
Identified set of relevant guidelines
Design of the GUI
Creation of the template and 9 mock-ups
Execution of the user study
Distribution and collection of the questionnaires
Conduction of the interviews
Analysis of the answers
Open and axial coding
Charts creation and results explanation
Analysis of the answers
Update and implementation of the GUI

Fig. 1. Research process.

the open codes “don’t show again button”, “example only if requested by user”, and “option to configure time frames for recommendations” to the “Configurability of the notification” axial code. Following the methodology described in [36], we looked for axial codes that either relate to the phenomena under study, causal conditions which lead to the occurrence of the phenomena, attributes of the context of the investigated phenomena, intervening conditions that influence the investigated phenomena, strategies that the actors use to handle the phenomena, and the consequences of their actions and interactions.

The codes were extracted by the second author and then reviewed by the first author of the paper. Whenever there was a disagreement on an assigned open code or on the subsequent grouping into an axial code, both researchers re-read the answer of the respondent and corrected the assigned code to what—according to both researchers—reflected the answer of the interview respondent the most.

As already mentioned, the main research contributions of this paper (guideline 4) are the development of a new GUI design for an IDE command recommender system and its evaluation. In our study, from the perspective of the adopted Design Science research paradigm, the research rigor (guideline 5) consisted in the identification and application of relevant GUI design guidelines during the construction of the artifact (Section 4.1), its evaluation (Section 5), the discussion of possible threats to validity, and in the devising of the tactics to mitigate the impact of the identified threats (Section 6).

This paper describes one iteration of a design search process (guideline 6) that started with a literature study, during which we identified relevant guidelines, later used for designing the GUI, followed by the evaluation of the mock-up with a questionnaire and semi-structured interviews, followed by the analysis of the evaluation results, based on which the GUI design was updated and finally implemented. The overall research process is also depicted in Fig. 1. This paper describes the execution of the first iteration and communicates the important results (guideline 7).

3. Proposed command recommendation GUI

In this section, we present a novel IDE command recommendation GUI. The structure of the proposed GUI can be split into three main parts. Part 1 contains the description of the suggested command: the command name, the shortcut, and a brief description of what the command does. Part 2 contains the explanation of why the command is recommended: the context in which it is often used, the proportion of similar users who already use this command, and a persuasive statement. Finally, Part 3 presents the effect of the command: two screenshots to graphically explain the consequences of the command execution.

Table 1
Guidelines for each GUI part (1. command name and description, 2. explanation, 3. usage example) and the supporting literature [19].

Guideline	Part		
	1	2	3
Support “understandability” [13]	×		×
Enable “transparency” [13]		×	
Support “assessability” [13]	×	×	×
Support “attribute-based” choice pattern [39]		×	
Support “socially-based” choice pattern [39]		×	
Support “experience-based” choice pattern [39]		×	
Support “consequence-based” choice pattern [39]			×
Adopt the “access information and experience” strategy [39]		×	×
Adopt the “represent the choice situation” strategy [39]		×	×
Adopt the “combine and compute” strategy [39]		×	
Adopt the “evaluate on behalf of the chooser” strategy [39]		×	
Use simple and natural language [40]	×		
Minimize user’s memory load [40]	×		
Provide shortcuts [40]	×		

Fig. 2 shows a sample of the proposed GUI for recommending IDE commands, with annotated parts. In the figure, the Open Resource command is suggested to the user. The rationale of the information included in each part is based on the GUI design guidelines presented in Section 4 and summarized in Table 1. In Section 4, we will further discuss the motivations for the proposed GUI in relation with the previously mentioned guidelines; here we discuss in detail the content of the GUI parts.

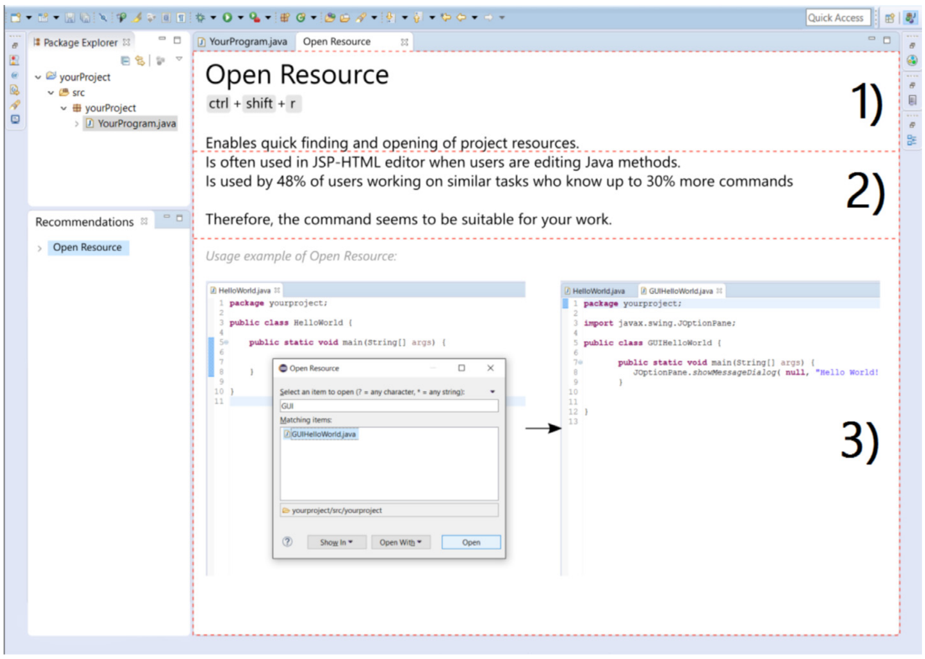
3.1. Command name and description

Part 1 starts with the command name, which is either the same as the one used in the Eclipse menu or it is extracted from the ending of the command unique identifier, as logged by Eclipse Usage Data Collector.[3] For instance, the identifier of the Open Resource command is “org.eclipse.ui.navigate.openResource”.

After the name, there is the command’s shortcut to enable more experienced users to execute it faster. In our study, we used the default shortcut of Eclipse version for Windows operating system. For example, the shortcut of the Open Resource command is “ctrl + shift + r”.

Beneath the shortcut, there is a brief, single sentence description of the command functionality, which includes the information about the benefit for the user. For instance, in Fig. 2, the Open Resource command “enables quick finding and opening of project resources”.

[3] http://www.eclipse.org/epp/usagedata.



Open Resource

ctrl + shift + r

Enables quick finding and opening of project resources.

Is often used in JSP-HTML editor when users are editing Java methods.
Is used by 48% of users working on similar tasks who know up to 30% more commands

Therefore, the command seems to be suitable for your work.

Usage example of Open Resource:

```
package yourproject;

public class HelloWorld {

    public static void main(String[] args) {


    }

}
```

```
package yourproject;

import javax.swing.JOptionPane;

public class GUIHelloWorld {

        public static void main(String[] args) {
            JOptionPane.showMessageDialog( null, "Hello World!
        }


}
```

Fig. 2. Command presentation GUI with indicated main parts [19].

3.2. Explanation

Part 2 contains the explanation why the command is recommended. The explanation is structured according to the *argumentative facts* explanation approach [37]: "A, B, therefore C".

The first sentence (i.e., A), describes 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. 2, the Open Resource command "is often used in JSP-HTML editor when users are editing Java methods".[4] This context, which can be identified by the recommender system, if it is context-aware, also represents the situations in which the recommendation recipient often works. For the purposes of this study, we manually identified the relevant context. In Section 7, we present an approach to automatically generate this sentence.

The second sentence (i.e., B) indicates the proportion of users who already use this command and are similar to the target user, or are slightly more proficient. For instance, in Fig. 2, the Open Resource command "is used by 48% of users working on similar tasks who know up to 30% more commands". We note that in this study, we arbitrarily defined the proportions.

The third sentence (i.e., C) of the recommendation explanation, "Therefore, the command seems to be suitable for your work", is the conclusion of the argument. 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, is implicitly urging the user to use the command.

3.3. Usage example

In Part 3, which is—as Part 1—dedicated to the command presentation, we placed a graphical usage example consisting of two screenshots, connected with an arrow. Even though studying the proposed example may be time consuming for the user, we decided to include it because assessing the recommendations of software engineering recommender systems typically requires a higher level of cognitive effort compared to recommender systems for other domains [13] and users are often willing to take their time for making good decisions [38].

The first image in the example shows a typical situation in which the proposed command can be used and the second shows how executing the command affects that situation. For the purposes of this study, we manually generated usage examples. In Section 7, we discuss how usage examples for IDE commands could be generated automatically.

4. Related work

In this section, we present some GUI design guidelines drawn from existing literature, which we used to create the proposed GUI for an IDE command recommender system. We also describe existing GUIs for command recommender systems, which were used in other types of high-functionality applications, and we compare them with our proposal.

4.1. Design guidelines

As already described in the previous section, the GUI design, which was evaluated in the user study reported in this paper, consists of three main parts: command name and description, explanation of recommendation, and command usage example. The rationale for this design is derived from a set of selected guidelines, which we identified in the literature. In Table 1, we show the mapping between the selected guidelines and the GUI parts. In this section, we provide a detailed overview of the relevant literature and describe how the selected guidelines were implemented in the proposed GUI.

Murphy-Hill and Murphy [13] identified five important success factors for the interface of a recommender system for software engineering:

- *understandability*: the degree to which a user is able to determine *what* a recommender system is suggesting;

[4] Parts of the context that do not apply are omitted. For instance, the preceding activity in Fig. 2.

- *transparency*: the degree to which a user is able to determine *why* the recommendation is being provided;
- *assessability*: the degree to which a user is able to assess whether or not a recommendation is relevant and is one that a user wants to take action upon;
- *trust*: the degree to which a user is able to trust the recommendation to benefit her in the way the system implies; and
- *distraction*: the degree to which a user is distracted by the recommendation.

Our proposed GUI addresses the *understandability, transparency*, and *assessability* factors directly. The single sentence description of the command functionality supports *understandability* and *assessability*, by telling the user what the recommended command does and what will she gain if she uses it. Part 2 of the GUI contains the explanation why the command is recommended, in order to enforce the *transparency* of the system. Additionally, the context description, which is a part of the explanation, also supports *assessability*, by referring to the situations in which the command is often executed. Finally, the *assessability* success factor is also supported by the usage example.

In Section 5.2, we present the opinions of the interviewees on how the users should be notified of a recommendation, which is related to the *distraction* factor. The *trust* factor is out of the scope of this study, because it is typically affected by different components of a system, not only by the GUI.

We are not aware of any other explicit application of Murphy-Hill and Murphy's guidelines in the context of recommender system GUIs.

Furthermore, the GUI we propose is influenced by the ASPECT and ARCADE models [39], which are important references for the recommender systems community when designing recommender system's interface. Based on an extensive study of the existing body of knowledge on how people make preferential choices, which mainly originates from the psychological research, Jameson et al. devised the ASPECT model. This is the first model that provides a synthesis of relevant theories on human decision making that are adapted to the needs of human–computer interaction [39]. The ASPECT model defines six basic patterns that people apply alternatively or in combination when taking everyday decisions:

- *attribute-based choice*: the choice occurs by choosing attributes that are relevant for the evaluation and by evaluating items based on the values of the chosen attributes;
- *consequence-based choice*: the choice occurs based on the expected consequence of choosing an item;
- *experience-based choice*: the choice occurs based on the previous experience, emotions, or habits;
- *socially-based choice*: the choice occurs based on the expected reactions of relevant people when choosing an item;
- *policy-based choice*: the choice occurs based on the applicable policy in the current context; and
- *trial and error-based choice*: the choice occurs based on a learning process in which an item is (randomly) chosen and the consequences of the choice are evaluated; if the outcome is not satisfactory, another item is chosen (also considering the experience made with the previous choices), and the process is repeated until the final choice is satisfactory.

Additionally, Jameson et al. studied various approaches that have been used by researchers to provide choice support. They devised the ARCADE model, which is a synthesis of high-level strategies that a system can use to support any of the choice patterns included in the ASPECT model and consequently can improve the quality of the recommender system users' decisions. The ARCADE model defines the following six strategies:

- *access information and experience*: the system provides to the chooser information that is relevant in the decision process;
- *represent the choice situation*: the system arranges the graphical content appropriately, in order to effectively support the interpretation, processing, and actions taken by the user;
- *combine and compute*: the system performs mechanical tasks on behalf of the user;
- *advise about processing*: the system (explicitly or implicitly, through an appropriate interaction design) suggests to the user which actions to take, before she takes the final decision;
- *design the domain*: the system represents the reality in a way that supports the decision process; and
- *evaluate on behalf of the chooser*: the systems makes a decision and (explicitly or implicitly) advises the user to accept it.

We applied four ARCADE strategies, namely, *access information and experience, represent the choice situation, combine and compute*, and *evaluate on behalf of the chooser*, to directly support *attribute-, socially-, experience-*, and *consequence-based* choice patterns, from the ASPECT model. With the context description that also represents the situations in which the recommendation recipient often works, we support the *experience-based* choice pattern, following the *represent the choice situation* strategy, by making the situation recognizably similar to the previous experiences of the user. With the sentence that indicates the proportion of the users who already use the recommended command, we support the *socially-based* choice pattern, following the *access information and experience* and the *combine and compute* strategies, by providing the information that emphasizes the sense of identification and affiliation of the user to a group of similar people. Moreover, by expressing the level of the popularity, we also support the *attribute-based* choice pattern. Furthermore, the last sentence in the explanation follows the *evaluate on behalf of the chooser* strategy, by implicitly urging the user to use the command. Finally, the usage example supports the *consequence-based* choice pattern, following the *access information and experience* and the *represent the choice situation* strategies, by providing the information about the initial state in the first image and "a preview of the experience of executing a particular action" [39, pp. 41] in the second image.

In the proposed GUI, we do not focus on the *trial-and-error* choice pattern, because the support for it is already provided in an IDE, by the Undo command. We also do not consider the *policy-based* pattern, since we do not think it is relevant in our domain.[5]

In addition to the guidelines in [13] and [39], we also followed some usability guidelines in a human–computer dialog, provided by Molich and Nielsen [40]. The entire list consists of the following guidelines:

- *use simple and natural language*;
- *speak user's language*;
- *minimize user's memory load*;
- *be consistent*;
- *provide feedback*;
- *provide clearly marked exits*;
- *provide shortcuts*;
- *provide good error messages*; and
- *prevent errors*.

In the proposed GUI, we have considered the guidelines that suggest to *use simple and natural language, minimize user's memory*

[5] Despite the fact that different policies, such as enforced coding styles, are not uncommon in software engineering, we are not aware of any policies or policy-makers that would define how the programmers should use an IDE on the level of the menu buttons or shortcuts that should or should not be used.

load, and *provide shortcuts*. The structure and the inclusion of the specific elements in the GUI—in particular in Part 1, which includes the name, the shortcut, and the description of the recommended command—is motivated by or designed in compliance with these three guidelines.

Because the population of IDE users is diverse and we do not know what specific language they would prefer in the GUI, we decided to use the terminology used in Eclipse IDE, even though it may violate the *speak user's language* guideline. Furthermore, as this paper is focused on the content of the command recommendation presentation, we consider the remaining guidelines out of scope since they are focused on the implementation of the system and on the human–computer interaction in a real-life setting.

Finally, we adopted the *argumentative facts* recommendation explanation approach, suggested by Zanker and Schoberegger [37]. In [37], the authors have studied different types of recommendation explanations and discovered that if the facts in the argument have an "A, B therefore C" structure, then the explanation—in our case, Part 2—is more persuasive.

4.2. Existing GUIs for recommending commands

Command recommender system GUIs that present IDE commands have been implemented in Spyglass [41], which recommends navigation commands in IBM Rational Team Concert,[6] and Vignelli [42], which detects design flaws in the code and recommends suitable refactoring commands in IntelliJ IDEA.[7] Since these recommender systems have been tailored to suggest specific types of commands, their GUI design cannot be directly applied to other types of IDE commands. For instance, in Spyglass, at the beginning of the recommendation presentation, there is a description of how the recommended command could have replaced the actions that the user recently performed. We assume that the Spyglass GUI design can be easily reused for presenting any IDE command that accelerates an inefficient sequence of user actions, such as editing commands that replace typing; however, this particular design cannot be reused for presenting the commands that provide functionality which can only be accessed in one way, for example by executing Debug, SVN Commit, or Open Project commands. Likewise, the GUI implemented in Vignelli cannot be generalized to all types of IDE commands because it first informs the user about the issues in the source code and then guides her through the steps required to fix them. Due to that, the Vignelli GUI design can only be applied to editing commands.

On the other hand, various systems providing command recommendations within other high-functionality applications have been developed: OWL [43] for Microsoft Word,[8] CommunityCommands [44] for AutoCAD,[9] and two recommender systems developed by Wiebe et al. [45] for GIMP, namely, *social* and *task-based* recommender systems. Even though these tools do not focus on IDE command recommendations, the criteria that an effective recommendation presentation must satisfy are similar.[10] We briefly discuss each of them in turn. In Table 2, we provide a comparison of the existing GUIs with the one we designed and evaluated.

In OWL, command recommendations must be requested by the user through interaction with the toolbar. Once requested, the recommendations are provided in a list, where each recommendation includes the name of a recommended command, a short description of the command, and a recommendation score. A user can click on the command name to access a help page about the command. The scores are used to express frequency of use: the first command in the list is always scored 100 and the score of the tenth command is always 50; the score of the other eight commands is "proportional to their spacing in the frequency of use list" [43, pp. 197].

In CommunityCommands, the recommendations are presented in a palette. Each recommended command is presented with a button on which the command name and a bar representing the relevance to the user's current workflow are displayed. Clicking the button executes the command and hovering over the button displays a tooltip with a short description of the command and personal, manager's and co-workers' notes on the command. Each command button also contains two additional buttons to pin the command to the list or to remove it from the list.

Wiebe et al. developed *social* and *task-based* command recommender systems for GIMP, using two approaches to present the recommendations: 1) a list of recommendations can be visualized in a palette with additional information displayed when the user selects a command; or 2) there is no list of recommendations with additional information displayed when the user hovers over the commands in the menu. In both recommender systems, the additional information contains the system's confidence in its recommendation and the description of the command, which is longer than in OWL and in CommunityCommands. The difference between these systems is that the *social* recommender system shows the percentage of all users using the command and the percentage of similar users using it, while the *task-based* recommender system shows a list of tasks related to this command. The authors do not report how similar users and related tasks were identified.

The main distinctive features of the GUI proposed in this paper 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.

5. Results

In this section, we present the results of the evaluation of the proposed GUI. As we mentioned earlier, the evaluation of the designed artifact consisted of two parts: we conducted a user study with a questionnaire, to assess the usability and acceptance of the GUI, as it is perceived by the target population, and we performed a set of interviews to collect the expectations of software developers about which data can and should be used by an IDE command recommender system to generate recommendations, what information should be included in the GUI, and how would they like to be notified about the recommendations. The collected data in its raw form is publicly available at: https://doi.org/10.5281/zenodo.581390 [46]. We report on the results of both parts in turn.

5.1. Questionnaire

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.

5.1.1. Q1: Which types of developers participated in the evaluation of the proposed GUI?

To answer Q1, we analyzed the answers from the first two parts of the questionnaire, which included the questions about

[6] http://www.ibm.com/software/products/en/rtc.

[7] https://www.jetbrains.com/idea.

[8] https://products.office.com/en-us/word.

[9] http://www.autodesk.com/products/autocad/overview.

[10] We note that according to Matejka et al. [44], there are two main strategies for recommending commands: *global* suggestions and *opportunistic* suggestions. In the first case, the system provides general recommendations, based on the entire command usage history; in the second case, the system mainly takes into account what the user is doing *at the moment* and provides recommendations that are particularly relevant in the current context. In this paper, we focus only on the first scenario, which is called *global*.

Table 2
Comparison of the existing GUIs with our proposal, according to the user interface elements that support understandability, transparency, and assessability [13].

System	Understandability	Transparency	Assessability
OWL [43]	• Single sentence description • Link to the Office Word Help page	• Rank of the command	• Single sentence description • Link to the Office Word Help page
CommunityCommands [44]	• Single sentence description		• Bar representing the relevance of the recommendation to the user's current workflow
GIMP *social* recommender system [45]	• Extensive description	• Percentage of all users using the command • Percentage of similar users using the command	• Extensive description
GIMP *task-based* recommender system [45]		• Extensive description	• Extensive description • List of related tasks
Our suggestion	• Single sentence description • Usage example	• Recommendation explanation	• Single sentence description • Recommendation explanation • Usage example

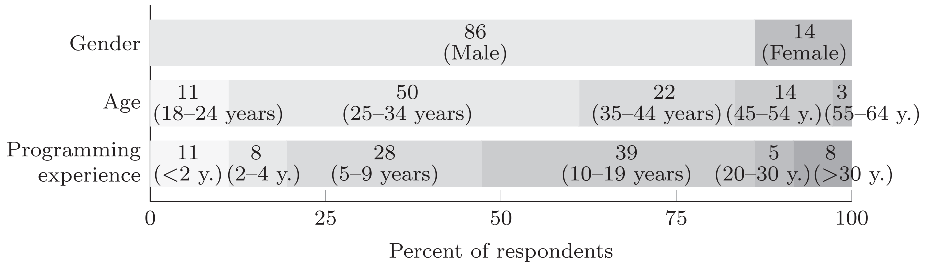


Fig. 3. Age, gender, and programming experience of the participants (Q1).

gender, age, years of programming experience, activities performed in an IDE, recently used IDEs, recently used programming languages, agreement with the statements about learning new IDE commands and using an IDE command recommender system, and previous usage and familiarity with the nine commands included in the questionnaire.

As can be seen in Fig. 3, which shows the demographic data of our sample, 14% of participants are female, the age range is between 18 and 64, and more than 80% of participants have at least 5 years of programming experience.

All except one participant—who was not using any IDE—are editing and navigating in an IDE, 89% of participants are also debugging, and 56% are using version control. 56% used Eclipse recently, which means that more than one half of the participants should be familiar with the environment in which the GUI mock-ups were presented. 78% were programming in Java recently, 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.

When asked about their general attitude towards IDE command recommendations, 83% of participants agree or strongly agree with the statement that it would be useful for them to learn new commands. 94% of participants think that it would be useful for others to learn more commands. 80% agree or strongly agree that an IDE command recommender system is a good idea, 17% are neutral, and only 3% disagree. Detailed results are shown in Fig. 4.

In general, the participants have a good knowledge of IDE commands: 33% of participants had previously used all nine commands included in the questionnaire and only 6% had 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”. Fig. 5 presents detailed results of the answers to those two questions, in respect to each of the nine commands included in the questionnaire.

In Fig. 6, you can see the exact distribution of the command selected in the third part of the questionnaire, in respect to the knowledge of the command and the previous usage. To answer the questions in the third part of the questionnaire, which are related to Q2, Q3, and Q4, and are discussed in the following sections, 22% of participants selected Breakpoint Properties command or Surround with Try-Catch, 19% selected Open Resource, 14% selected Rename Element, 8% selected Next Editor, 6% selected Navigate Back or Step Return, 3% selected Organize Imports, and none selected Step Return. Only 53% of participants selected a command they have never used before.

When we observe the familiarity levels, we see that 36% of participants selected a command of which they were unaware, 14% selected a command about which they had some idea, 17% selected a command about which they had a clear idea, and 33% selected a command that they could explain.

5.1.2. Q2: How well do participants accept IDE command recommendations, if they are presented by the proposed GUI?

Four statements included in the questionnaire correspond to Q2. The participants had to express their agreement with “I would find the command easy to use”, “I would imagine that most people would learn to use this command very quickly”, “I would find the command useful in my job”, and “I intend to use the command in the future” statements.

As we report in Fig. 7, 86% (19%+67%) of participants perceive the recommended command as easy to use and the majority, 58% (14%+44%), also think that other people would quickly learn how to use this command. Furthermore, the acceptance of the recommendations was high as well: 77% (19%+58%) of the participants would find the command useful in their job and 64% (25%+39%) plan to use it in the future.

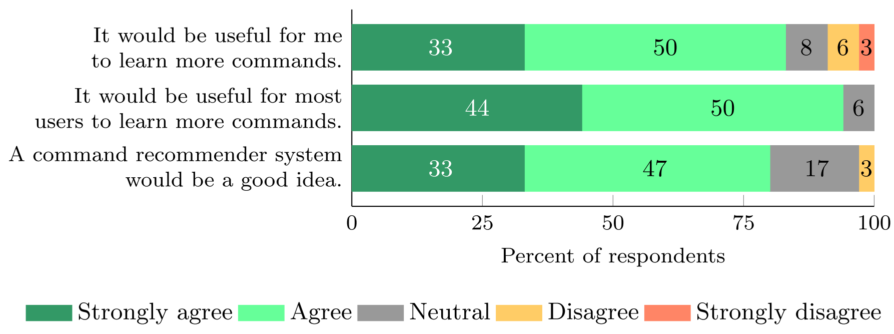


Fig. 4. General attitude of the participants towards the concept of an IDE command recommender system (Q1).

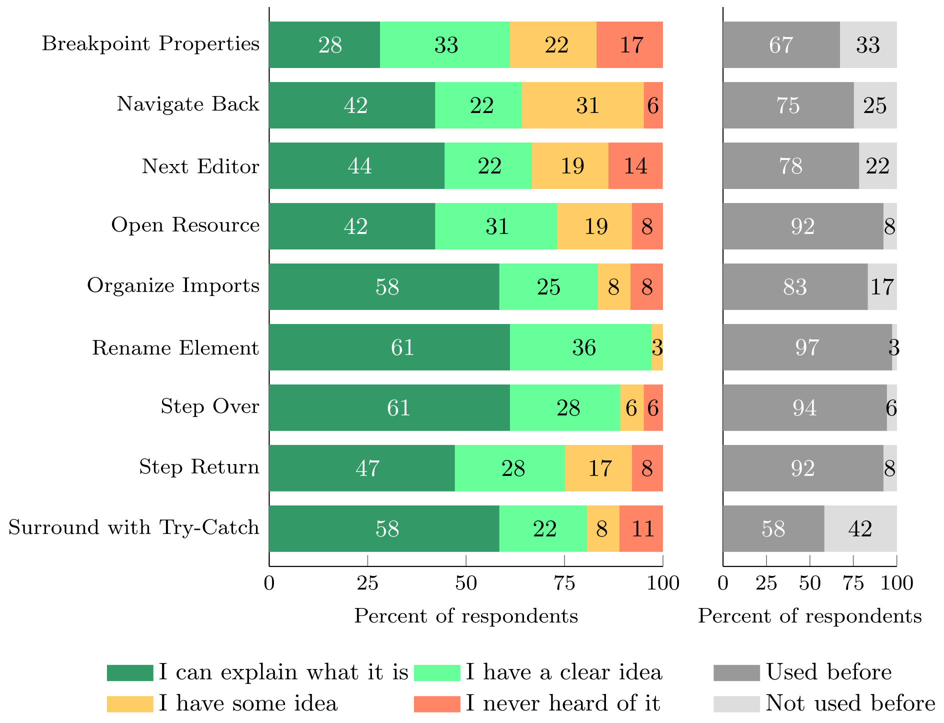


Fig. 5. Familiarity and previous usage of the nine commands included in the questionnaire (Q1).

5.1.3. Q3: How well is the proposed GUI accepted by the participants?

To answer Q3, we studied the replies to two statements and two questions, namely: “I think I would need some additional support to actually learn how to use the command”, “I find the command presentation unnecessarily complex”, “Do you think that some information is missing in the command presentation?”, and “If yes, which?”

There is a disagreement between the participants about the information that has to be provided by the GUI. As can be seen in Fig. 8, 36% (8%+28%) think that they would still need some additional support to learn how to use a command presented by the GUI and 39% (8%+31%) think they would not. We note that the need for an additional support varies considerably between the commands, as can be seen in Fig. 9.

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.

5.1.4. Q4: How useful do the participants perceive the parts of the designed GUI?

To answer Q4, we observed which of the three parts of the GUI the participants marked as necessary and/or useful for evaluating the quality of the command recommendation.

As can be seen in Fig. 10, the command description and the usage example were perceived as useful by 89% of the participants, but only 53% of the participants 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% of the participants and the usage example by 75%, but the explanation is perceived necessary only by 36% of the participants.

5.1.5. Potential correlations

In this section, we list our observations that might be useful for other researchers. However, because our sample was relatively

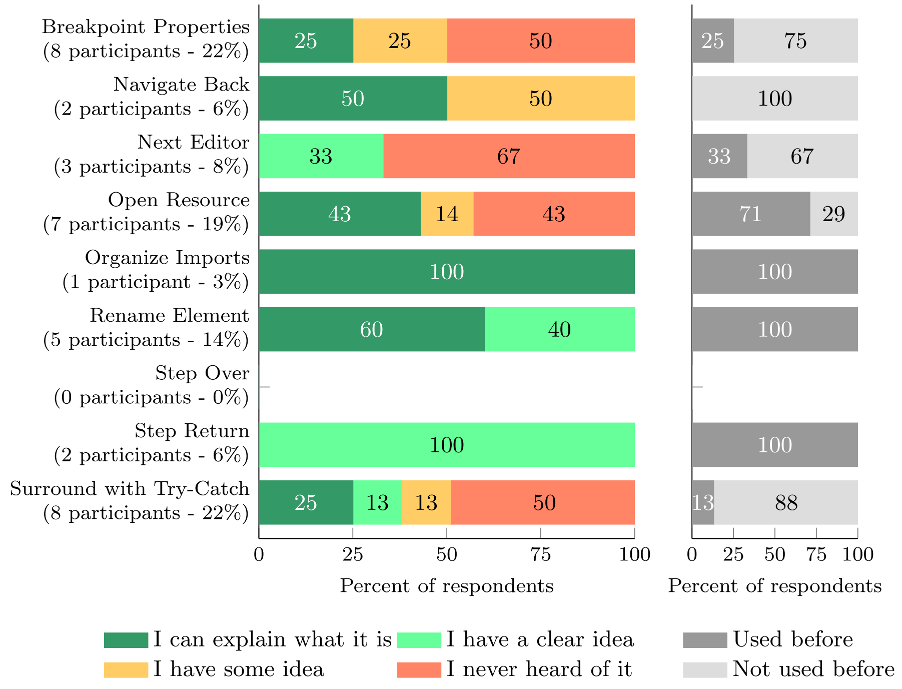


Fig. 6. Familiarity and previous usage of the commands selected in the third part of the questionnaire (Q1, Q2, Q3, Q4).

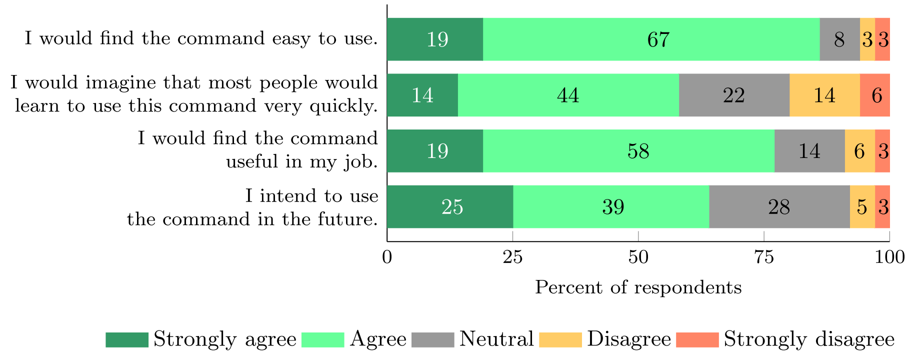


Fig. 7. Results of the agreement with the statements about the command ease of use, its usefulness, and the intention to use it in the future (Q2).

commands, as can be seen in Figure 9.

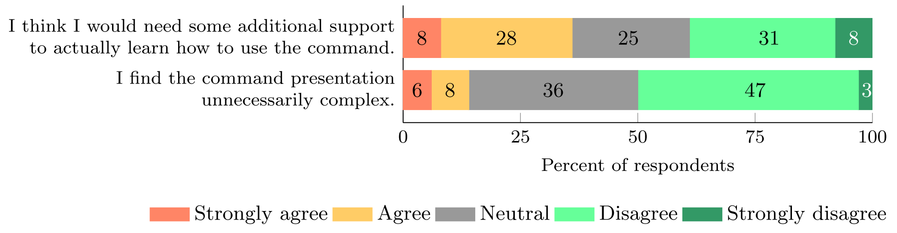


Fig. 8. Results of the agreement with the statements about the command presentation (Q3).

small, the power of the statistical tests we performed to identify the correlations between the responses—which are not self-evident—is low. The exact data is reported in Table B4, which appears in Appendix.

We notice a high interest among younger participants to learn new IDE commands, while older participants tend to be more skeptical. Interestingly, we do not detect such a pattern with respect to the programming experience.

Those participants who find the command easy to use, also tend to find the presentation less complex. Moreover, older participants are more likely to perceive the presentation as unnecessarily complex than younger, regardless of their programming experience.

Finally, those participants who think that they should learn more commands also perceive the recommendations as more useful, compared to the other participants.

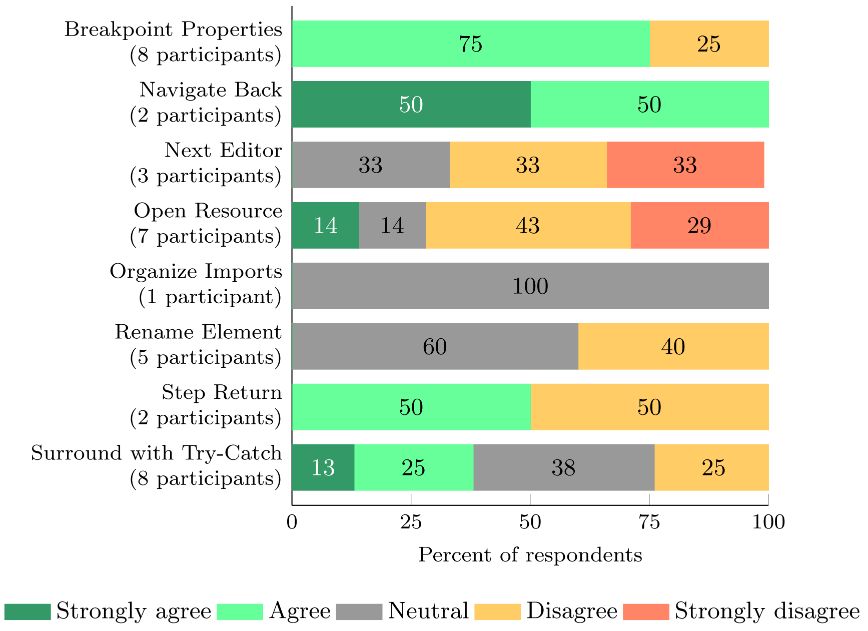


Fig. 9. Agreement with the statement: "I think that I would need some additional support to actually learn how to use the command", per selected command (Q3).

5.2. Interviews

We interviewed 11 software developers, who agreed to be interviewed after we invited them personally, during the XP 2016 conference. We recorded and transcribed their answers to the semi-structured interview questions, from which we extracted 161 open codes during *open* coding and we grouped them using *axial* coding [35] into 13 axial codes.

Interviews were conducted to compare, explain, and extend the results of the questionnaire. The identified axial codes can be considered possible requirements for an IDE command recommender system. We grouped these codes according to the research question they address (see Section 2).

5.2.1. Q5: Which types of developers participated in the interviews?

Out of 11 interviewees, 8 think that they have to learn more IDE commands and would use an IDE command recommender system if it was available. Two interviewees are using IntelliJ IDEA, which already includes a command recommender system that they are using and that they believe is sufficient. One interviewee said: "I am at the point where I wish that the tool would do more for me", thus, he would generally be in favor of using a recommender system, but he is skeptical of whether the recommendations could be accurate enough.

Almost all interviewees currently work in industry or have worked in industry in the past. Two have more than 30 years of software development experience, four have between 10 and 20, three have a few years of industrial experience, and two have less than two years of software development experience (one of them only in the academic environment).

Six of the interviewees told us during the interview that they filled in the questionnaire before participating in the interview.

5.2.2. Q6: Based on which data should the recommendation algorithm work to provide useful recommendations?

We extracted the following 7 axial codes:

1. *Commands frequently used by others*: the recommendation algorithm should suggest commands that are frequently used by other developers (expressed by 8 interviewees). Respondents proposing this recommendation strategy mentioned that they find it interesting to learn from others and that the popularity of a command is a good justification for suggesting it. Three respondents said that they do not care about how often a certain command is used by others.
   Some example quotes are:
   - "I try to use anything that I see other people do, that I feel is useful."
   - "Periodically show a demo of how to use commands that the user never uses and a million of other people use, you know like in Amazon."
2. *Commands based on the IDE interaction history*: the recommendation algorithm should suggest commands that are based on the common working tasks of the current user (expressed by 3 interviewees).
   Some example quotes are:
   - "A system able to identify that I am doing something using several clicks that I could do with one click or a keyboard shortcut."
   - "I think it would be wise if it studied my pattern, like what I do more, and then try to find related commands of what I am doing."
3. *Commands based on intention*: the recommendation algorithm should suggest commands that are based on what the user wants to achieve. The recommender system should be able to "detect the intention" of the programmer and suggest commands that are useful in that specific context (expressed by 9 interviewees). Three respondents that suggest this recommendation strategy disapprove unrelated recommendations like the "Tip of the day", two approve of such recommendations.
   Some example quotes are:
   - "To be honest, if there is a shorter way to do something that I want to do, I would like the system to tell me how."
   - "If that could detect your intention before you actually do it that would be amazing."
4. *Commands that are more efficient than the ones executed by the user*: when the user executes one or more commands and there is a faster way to achieve the same objective, the recommendation algorithm should suggest how to do that (expressed by 2

interviewees). One example is to suggest shortcuts for the commands executed immediately before the recommendation.
Some example quotes are:
- "A system able to identify that I am doing something using several clicks that I could do with one click or keyboard shortcut."
- "I would be particularly interested in recommendations of shortcuts of commands."

5. *Commands that are new*: the recommendation algorithm should suggest commands that are unknown (expressed by 2 interviewees).
An example quote is: "If you've got something that pops up and says: 'You had never used this command and this would be useful. Are you interested?' and I could answer yes or no, then that might be interesting."
6. *Willingness to share meta-data about coding activities*: on one hand, most respondents are willing to install an IDE command recommender system that logs executed commands, collects meta-data about the edited source code, and submits this data to a central server, particularly if no personal data is collected (expressed by 8 interviewees). On the other hand, some believe that the companies for which they are working will not approve or they consider sharing the source code in which they are working on as problematic (expressed by 5 interviewees). They suggest to have the possibility to configure the level of how much data is shared when creating a new project (expressed by 2 interviewees).
Some example quotes are:
   - "Sharing data about how I am using the IDE and how many times I am repeating the same steps to get the result that I wanted, these things I might share."
   - "It depends on the company but if it was a personal project I wouldn't mind."
7. *Configurability of the recommendation algorithm*: the types of commands that are recommended should be based on the experience of the user (expressed by 1 interviewee). Moreover, it should be possible to disable particular recommendations (expressed by 1 interviewee).
An example quote is: "Maybe an option to disable that particular suggestion or set of suggestions for a certain amount of time."

5.2.3. Q7: Which information about a recommended command is expected?

We extracted the following 3 axial codes:

1. *Contents of recommendation*: the recommendation should contain the name and the shortcut of the recommended command (expressed by 9 interviewees) and explain the effect of the command through text, pictures, videos, or a preview on the actual code (expressed by 7 interviewees). Some respondents found that the explanation why a recommendation was generated is useful (expressed by 3 interviewees), one of those explicitly stated that it might increase the acceptance of the recommendation. One interviewee stated that he does not need that the system tells him why a recommendation is relevant. Some think that a long description of the command is not important (expressed by 2 interviewees).
Some example quotes are:
   - "I think that maybe just the name and shortcut in a toolbar that I can check while I am coding."
   - "I would say the name, the shortcut for execution and a description with pictures, maybe a link to a video."
2. *Recommendation should be as simple as possible*: some respondents are concerned that a recommendation GUI with description, example, and explanation requires too much effort to be understood (expressed by 4 interviewees). They suggest to explain the command with short descriptions, minimize the amount of occupied screen space or to only gradually add information, e.g., providing a usage example only if the user performs an "undo" after trying out the command, or that the recommendations should become less detailed over time.
Some example quotes are:
   - "The first time you ever hear about a functionality, then you need more context, but when you are really working with this thing everyday, then that kind of gets annoying. I don't need to read that description every time, over and over."
   - "I agree, I won't be annoyed as long as it does not take half of the screen or more and does not distract me."
3. *Configurability of the provided information*: the user should be allowed to choose which elements he or she wants to see in the recommendation GUI (expressed by 3 interviewees).
An example quote is: "Yeah, if I have used it and it worked, I wouldn't need an example anyway."

5.2.4. Q8: How should a recommendation be notified to the user so that the user keeps using the system?

We extracted the following 3 axial codes:

1. *Level of distraction*: the notification that a new recommendation is available should generally not force the user to interrupt his or her work (expressed by 7 interviewees). A recommendation can be checked when it appears or later when the user has time. For example, the IDE users could be notified about new recommendations via:
   - "light bulb icon" appearing within the code editor: the user is informed that a recommendation for the specific code is available when a light bulb (or something similar like a different coloring of code) appears next to the code it concerns, which the user can ignore or consult, by clicking on the light bulb icon (expressed by 4 interviewees);
   - notification similar to the notification of a new email (expressed by 1 interviewee): the user gets informed that a new recommendation is available; or
   - window containing the recommendations, which is always visible on the side (expressed by 2 interviewees): recommendations are visualized in a window on the side, which the user can consult when interested.

   If the system detects that the user is doing something that can have bad consequences for him or her, it should interrupt his or her work. For example, when the user is performing changes that cannot be undone. Furthermore, some interviewees find it acceptable to be interrupted regularly to learn a new command or to improve their working habits (expressed by 8 interviewees).
Some example quotes are:
   - "If in that case the pop-up told me to do it in a faster way with the shortcut, I wouldn't mind the interruption."
   - "What I really like in the way that IntelliJ does it, is that you put your cursor somewhere and you click on the light bulb and get the suggestion. I can click on it when I need it, and it does not occupy any screen space when I don't need it."
2. *Updating of recommendations*: while the user is working, the recommendations should be constantly updated (expressed by 1 interviewee).
An example quote is: "I think it should be always present and changing recommendations."
3. *Configurability of the notification*: the way a user is notified about a recommendation should be configurable, for example, the user should be allowed to choose the level of intrusiveness, the time interval between recommendations, or the recommen-

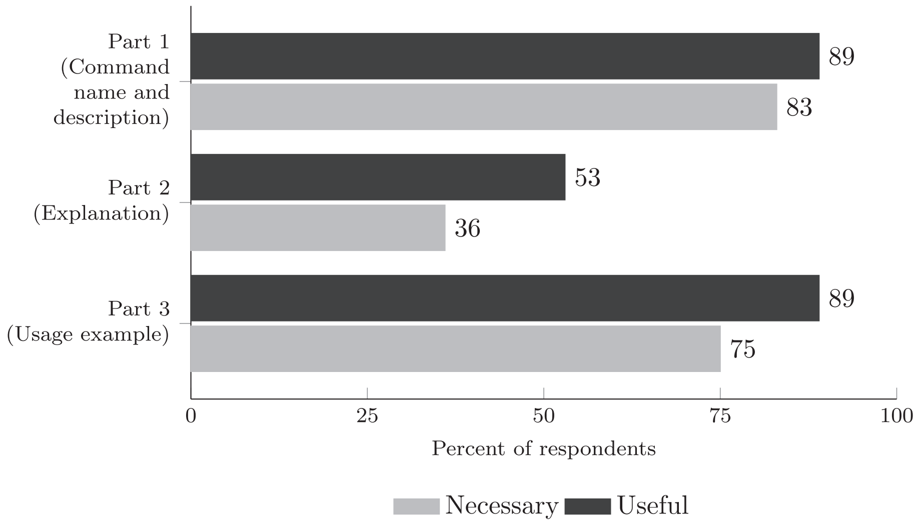


Fig. 10. Results of the questions about the usefulness of the GUI parts (Q4).

dation GUI should contain a “Do not show this again” button (expressed by 3 interviewees).

Some example quotes are:

- “Then it might do that again for like five times, and allow me to close it with a ‘don’t show again’ option.”
- “Maybe an option to disable that particular suggestion or set of suggestions for a certain amount of time.”

5.2.5. *Summary*

The analysis of the interviews revealed some important aspects that could influence the acceptance and the viability of the GUI in a real-life setting. We see that some interviewees would like to have more control over the types of recommendations that are shown, not only over the information that is visualized. Moreover, they would like to control the types of data that are shared with the recommender system (if possible, on the project level). Certain interviewees believe that good command recommendations are those that are popular, while others prefer contextualized recommendations. In the GUI, they would mostly like to see the information about the command, such as: what the command does, how it is used, what will be an immediate effect of the command, etc. Some participants think that over time the amount of information included in the GUI should decrease.

If we compare the results of the questionnaire with the results of the individual interviews, we can confirm the positive attitude towards an IDE command recommender system, as reported in Fig. 4. Also the participants to the interviews are interested in such a system, they are willing to share the collected data (if their organization approves it), and they are interested in command recommendations based on the popularity of others, their own IDE interaction history, or based on the task they currently try to perform. Their motivation is to become more efficient in their work and to learn something new.

When asked about how the recommendation should be presented, the respondents also confirmed the results of the questionnaire about the need and usefulness of the various parts of the proposed GUI. Similarly to the results presented in Fig. 10, the name and the description of the recommended command are considered necessary and useful. Furthermore, usage examples, presented with pictures, videos, or a preview of the executed command within the user’s source code, are considered useful; however, even though the participants are interested in examples, some suggested to omit them, especially when the user becomes more experienced, in order to reduce the amount of space that the GUI occupies on the screen. And finally, the explanation of the recommendation—as also emerged from the results of the questionnaire—was not perceived as useful by all the interviewees: some consider it useful, some assume that it might be useful for others, and some do not consider it useful at all.

To summarize, the answers provided by the interviewees are aligned with the answers to the questionnaire, and they reveal additional insights of the expectations and priorities that the respondents have about the GUI for presenting IDE command recommendations.

6. Discussion

In this section, we discuss the results of the evaluation of the designed GUI, as well as the validity threats. Furthermore, we present an updated version of the GUI, modified based on the evaluation results.

6.1. *Evaluation results*

Our results show that the majority of the software developers participating in the study want to learn more IDE commands and they have a positive attitude towards an IDE command recommender system. Moreover, those who think that they should learn more commands also tend to find the recommendations more useful, which is encouraging, since these are our target users.

Nevertheless, as we discovered through the analysis of the interviews, the actual acceptance of the recommender system in practice may be easily jeopardised. For instance, the user may be distracted by the notification that a recommendation is available or she may not be able to configure the shared data, which may prevent the usage of a recommender system at the work place.

According to Gedikli et al. [38], recommendation presentation is efficient when it reduces the cognitive effort required in the decision process. Based on the answers related to the ease of use of the command itself, we can assume that the suggested GUI is relatively efficient. However, the answers related to the statements about the need for additional support, presentation complexity, and the desired additional information indicate that the GUI can be further improved.

During the interviews, some respondents suggested to simplify the GUI even more, to show certain elements only in specific situations, or to hide certain elements over time, so that the GUI reduces to the mere description of the command, once a user is familiar with this command. The interviewed software developers seem to prefer short "hints" over lengthy detailed recommendations, because that would reduce their effort in understanding the recommendation.

Since the perceived complexity and the need for additional information vary between the commands and between the users, we assume that an effective solution should also support the adaptation of the GUI, based on the user's profile and based on the command. However, particularly when it comes to the modifications of the GUI based on the commands, we also have to consider the consistency principle [40].

Some interviewees suggested to provide a step-by-step video guide. In that way, for the easy commands, the video would probably be very short and often unnecessary, and for the complex commands, we can provide a lot of additional information, without changing the basic structure of the GUI.

The most unexpected finding of our study is that the recommendation explanation is perceived considerably less valuable than the other parts of the GUI. This finding is surprising because it contradicts previous results of the research on general recommender systems: Herlocker et al. [47] showed that providing explanations in a movie recommender system improves the acceptance of the recommendations and Sinha and Swearingen [48] showed that the users of music recommender systems prefer the recommendations that they perceive as transparent. Nevertheless, our observations confirm the observations of Viriyakattiyaporn and Murphy [41] who also developed a GUI for recommending IDE commands and noticed that many study participants did not read the recommendation rationale. Thus, we can conclude that in the domain of IDE command recommender systems, the *transparency* factor [13] is not as important as the *understandability* and *assessability* factors [13] are.

6.2. Validity threats

To discuss validity threats, we use the list of threats defined by Wohlin et al. [49] and Yin [50], which include conclusion, internal, construct, external, and reliability validity threats. Together with the threats, we also define the tactics to minimize their impact [50].

6.2.1. Conclusion validity

Conclusion validity represents the reliability of conclusions drawn from the collected data [49]. In our case, the number of the participants in the user study was relatively small (36).

To reduce the threats to conclusion validity, we used *multiple sources of evidence* [50], i.e., we conducted also 11 semi-structured interviews with open-ended questions to better interpret the answers given to the questionnaire and to better understand the expectations towards the reasoning to generate and deliver recommendations.

6.2.2. Internal validity

Internal validity threats are related to possible wrong conclusions about causal relationships between treatment and outcome [49]. In our case, the main internal validity threat is the selection of the volunteers to participate in the study, since the volunteers are generally more motivated than the whole population, hence, the selected group is less representative. It is possible that the acceptance of the designed GUI is high because the questionnaires were answered by enthusiasts.

We aimed to decrease this threat by convincing also less interested and more skeptical XP 2016 participants to answer the questionnaire or to participate in the interview.

6.2.3. Construct validity

Construct validity is related to the generalization of the result to the concept or theory behind the study [49].

The designed GUI relies on the identified design guidelines, which exposes it to the risk that we might have ignored relevant literature. To reduce this risk, we relied on the most well-known authors in the field of recommender systems for software engineering, as well as the main references of the recommender system community for providing choice support in a human–computer interaction.

To decrease a potential *mono-method bias* [49], induced by the user study, we used open-ended questions in the interviews that were in part similar to the ones asked in the questionnaire. Consequently, we exposed our studies to the potential *interaction of different treatments*, by involving some participants in both of them. We addressed this threat keeping track of which interviewees filled in the questionnaire, before they conducted the interview and which not. We did not detect any important differences between the two groups.

6.2.4. External validity

External validity threats are related to the ability to generalize the results of the experiment to industrial practice [49]. In our case, both studies are largely exposed to this threat, since they were not conducted in a real-life environment; instead, the participants had to imagine hypothetical scenarios. Thus, it is possible that the perception of the proposed GUI would be significantly different if it was implemented in the IDE and the recommendations were presented during an average working day. Moreover, it is hard to assess how representative the sample of our participants is, in respect to the population of IDE users.

To reduce the threat of external validity of our research, we *used theory* [50] to support our decisions when building the GUI, i.e., we based the design on the guidelines identified in the literature.

6.2.5. Reliability

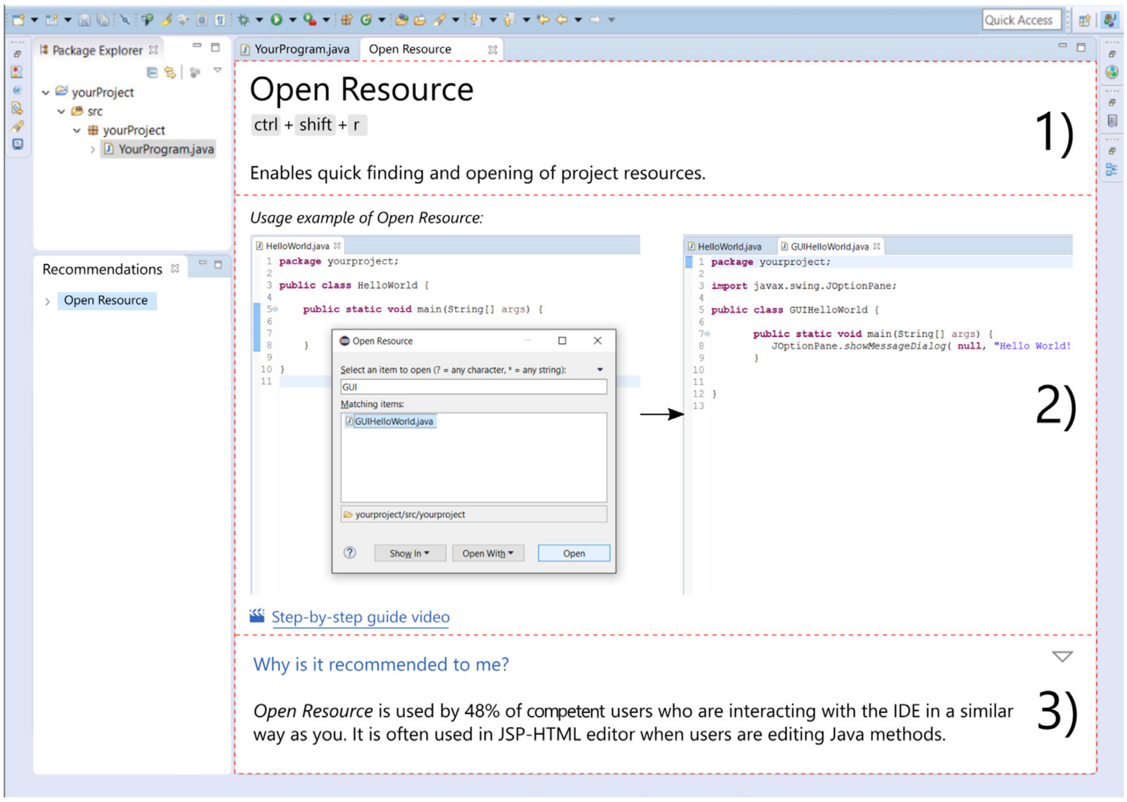
Reliability is concerned with demonstrating that a study can be repeated obtaining the same results [50]. In this context, it is important to document the procedures for the data collection [50]. To ensure reliability, this paper includes detailed description of the used evaluation instruments.

6.3. Updated version of the GUI

Based on the results of the questionnaire and the analysis of the interviews, we made a number of changes to the proposed GUI. The updated GUI is presented in Fig. 11.

At the end of the usage example, we now provide a link to a step-by-step video guide presenting the command usage, to offer clearer and more detailed instructions to those who need them. We placed the example after the command description and moved the recommendation explanation to the end, since it was perceived as less useful, compared to the other parts.

Furthermore, we modified the explanation text, because some perceived it as complex, not so useful, and—according to the side note on one questionnaire—even offensive. We replaced the sentence: "Is used by 48% of users working on similar tasks who know up to 30% more commands", with: "Is used by 48% of users who are interacting with the IDE in a similar way as you." If the recommendation algorithm is able to classify the current user as a novice, competent, proficient, expert, or master (we use the acquisition level classification proposed by Dreyfus and Dreyfus [51]), we suggest to mention only the percentage of the users who are one level more proficient than the target user, hence, implicitly encouraging the user to reach such a level. For example, if the user



Open Resource

ctrl + shift + r

Enables quick finding and opening of project resources.

Usage example of Open Resource:

HelloWorld.java

```
package yourproject;

public class HelloWorld {

    public static void main(String[] args) {

    }

}
```

Open Resource

Select an item to open (? = any character, * = any string):

GUI

Matching items:

GUIHelloWorld.java

yourproject/src/yourproject

Show In ▾ | Open With ▾ | Open

HelloWorld.java | GUIHelloWorld.java

```
package yourproject;

import javax.swing.JOptionPane;

public class GUIHelloWorld {

        public static void main(String[] args) {
            JOptionPane.showMessageDialog( null, "Hello World!
        }


}
```

Step-by-step guide video

Why is it recommended to me?

Open Resource is used by 48% of competent users who are interacting with the IDE in a similar way as you. It is often used in JSP-HTML editor when users are editing Java methods.

Fig. 11. Updated command presentation GUI [19].

is a novice, the explanation argument will be: “Is used by 48% of competent users who are interacting with the IDE in a similar way as you.”

We changed the order of sentences describing the context and command popularity, to indicate that this context is not predefined, but detected.

Finally, we eliminated the last sentence: “Therefore, the command seems to be suitable for your work”, and we introduced the explanation with a new statement: “Why is it recommended to me?” The explanation text can be minimized so that the users who find it less useful are able to hide it.

7. Putting the GUI into practice

In the future, we will evaluate the improved version of our GUI in a set of case studies within a business environment, using a recommendation algorithm, and measuring the user response to the recommendation while using the IDE. Currently, we are performing a study with novices, who are studying computer science at the Free University of Bozen-Bolzano. For that reason, we generated GUI content for a large set of recommendable commands. We observe that personalized recommendation explanations can be generated automatically without difficulty, while some challenges remain for other GUI components.

The content that we need to generate for Part 1 of the updated GUI consists of the command name, shortcut, and description. One can extract command names from their identifiers and the shortcuts from the keystrokes, which can be monitored automatically and mapped to the commands. The main challenge is the generation of command descriptions, which we created manually. To make the process more efficient, we believe that one possibility would be to extract command descriptions from web documentation, as Khan et al. [52] did for generating *task-related* recommendations in GIMP. However, we are not aware of any existing documentation mining technique that could be directly applied for generating IDE command descriptions, thus, we consider the future research on this topic as promising and potentially highly valuable.

To generate videos and usage examples that are presented in Part 2 of the updated GUI, one can follow the approach suggested by Murphy-Hill [2], who envisaged a monitoring tool that would capture the information about the development tools usage—in our case, IDE commands—and would also record the screen. From the videos of the software developers who are willing to share this data, the relevant parts of the recording could be extracted as suggested by Ponzanelli et al. [53]. We would like to implement this feature in the future, especially because the manual generation of videos and examples for the purposes of our study was extremely time consuming.

Unlike the first two parts of the new GUI, Part 3, which contains the explanation of the recommendation rationale, is expected to be personalized. Consequently, the content does not only depend on the recommended command, but also on the recommendation recipient. The suggested command explanations consist of two statements: one describing the command popularity and another describing the context in which the command is often used.

To automatically generate the text characterizing the context in which the command is often executed and in which the user often works, one can use six contextual factors that are included in the model defined in [27], namely: *current* and *previous activity*, *type*, *length*, and *complexity of the artifact under development*, and *user interface element with focus*. We suggest these factors because they explain what the developers usually do before and when they execute the recommended command, with which project artifacts they interact at this time, and which part of the IDE is used for that. We consider such information as simple to understand and highly informative. Then, one can calculate Term Frequency Inverse Document Frequency (TF-IDF) [54] score for each contextual factor value, by considering commands as documents and contextual factor values as terms. TF-IDF calculation is a classical approach to weight the importance of terms that exploits two ideas: frequent terms in a document are more important and rare terms in the collection are more important, therefore, one can assume that the contextual values with the highest scores well describe the most important contexts. To understand whether the recommendation recipient is familiar with the contextual factor values that have the highest TF-IDF scores, one can calculate the percentage of time she has been working, i.e., executed IDE commands, in the context described by a particular contextual value.

We note that we have already implemented the described approach in the above mentioned case study with novice programmers and we added the information about the command popularity, amongst the study participants. Five examples of generated explanations are listed below:

- "Force Return is used by 3% of users. It is often used when users are debugging a short Java class in Compilation Unit Editor".
- "Toggle Link With Editor is used by 8% of users. It is often used when users are working on a short Java class after editing".
- "Select Previous Word is used by 12% of users. It is often used when users are working on a short Java file in Compilation Unit Editor".
- "Rename Element is used by 26% of users. It is often used when users are editing".
- "Open Local File is used by 40% of users. It is often used when users are navigating to a short Java class in Package Explorer".

Based on the road-map presented in this section, which illustrates an approach for the automated generation of the GUI content, we can conclude that we will soon be able to fully automate the generation of the content for presenting IDE commands, which will ease the introduction of IDE command recommender systems in practice.

8. Conclusions

This paper describes the first GUI explicitly designed for recommending different types of IDE commands. State of the art GUIs for recommending IDE commands have been tailored to suggest only specific command types, such as commands for navigation [41] and refactoring [42]. Other GUIs have been designed for recommending commands in other high-functionality applications, such as Microsoft Word [43], AutoCAD [44], and GIMP [45]. By using our generic GUI, researchers working on IDE command recommender systems will be able to test their recommendation techniques in a real-life setting, without any human intervention; which was not the case in [14], for example. In particular, our results are useful for better understanding of the so-called *global* recommender systems, which generate recommendations that are based on the entire IDE command usage history and tailored to fit long term information needs of the users.

The design of the proposed GUI was motivated by a set of guidelines that we identified in the literature. It consist of three parts: command name and description, explanation of recommendation, and command usage example. To evaluate the effectiveness of this design, we analyzed the user acceptance and the perceived usability of the GUI in a user study that involved 36 software developers and we interviewed 11 developers, using semi-structured open-ended questions.

The results of our studies show that our GUI is well accepted by the study participants. The majority of them perceived the presented commands as easy to use and useful. Nevertheless, around one third of the participants believe that they would need additional information to learn how to use the recommended command; and from the interviews we learned that they would prefer to see more examples, for instance, in a video tutorial. On the other hand, some user study participants perceive the proposed GUI as unnecessarily complex; and from the interviews we learned that some developers think that the information provided should be reduced, once the recommendation recipient becomes more familiar with the system. Additionally, we discovered that the command presentation is perceived more useful than the explanation of the recommendation, which is in a disagreement with previous findings about the role of explanations in recommender systems.

We stress that the answers to the questionnaire used in the user study are aligned with the answers of the interviewees.

Based on the experimental results, we identified some potential improvements of the proposed GUI and we presented the steps that we consider necessary for the fully automated generation of IDE command recommendation presentations. Complete automation is important because it will enable the set of recommendable commands to grow together with the IDE. Hence, we invite other scientists to further explore the research problems addressed in this paper, such as automated generation of usage examples and extraction of command description from IDE documentation or from other publicly available sources.

In addition, there are some challenges that are out of scope of this paper, but are related to the improvement of the software developers' knowledge of available and potentially useful IDE functionality. Primarily, we expect that the proposed GUI will be integrated with existing and with novel recommendation algorithms. While a number of algorithms is already available [14–16], the unique live user study showed that the acceptance rate of the recommendations is not very high. Hence, the next challenge that we see is the improvement of the quality of the recommendation algorithms. One positive aspect of the proposed GUI is that it is independent from the underlying recommendation algorithm, hence, it can be reused by researchers focusing only on core recommendation algorithms.

Other challenges for future research are related to the timing [55] and the frequency [56] of command recommendations. One question is whether the IDE commands should only be recommended in the situations in which they are instantly executable? For instance, in AutoCAD, according to Li et al. [56], the users prefer more frequent recommendations, but they are still inclined to interact with the recommender system in *sessions*, when they open and inspect several different recommendations in a short time period [56]. Another question is how harmful (unnecessary) interruptions may be, since software development tends to require significant cognitive effort [14]. And finally, how should the system acquire user's attention? Which is the question that was partially addressed also in this paper, during the interviews.

This paper shows that a convenient GUI is critical to achieve high acceptance of IDE command recommendations. We hope that other researchers will make use of our results and will continue to investigate how to improve software developer's interaction with an IDE.

Appendix A. List of questionnaire questions

Table A3
Questions asked within the questionnaire.

No.	Question	Question type	Required	GQM-Question
1	Email (to receive feedback)	Open-ended		
2	Gender	Closed-ended, with the following single answer possibilities: “Male”, “Female”		1
3	Age	Closed-ended, with the following single answer possibilities: “<18”, “18–24”, “25–34”, “35–44”, “45–54”, “55–64”, and “>64”		1
4	Years of programming experience	Closed-ended, with the following single answer possibilities: “<2”, “2–4”, “5–9”, “10–19”, “20–30”, “>30”	×	1
5	Mark all the activities you perform in an IDE	Closed-ended, with the following multiple answer possibilities: “Editing source code”, “Navigating over source code”, “Debugging source code”, “Using version control”, “Other”. When choosing “Other”, the participant could answer providing a short description.	×	1
6	List the integrated development environments in which you were programming in the last two years (such as: Eclipse, VisualStudio, IntelliJ IDEA...)	Open-ended	×	1
7	List the languages in which you were programming in the last two years (such as: Java, C#, Ruby, Python, Cobol...)	Open-ended	×	1
8	It would be useful for me to learn more commands available in my IDE.	Closed-ended, five point Likert-Scale with the following answer possibilities: “Strongly agree”, “Agree”, “Neutral”, “Disagree”, or “Strongly disagree”	×	1
9	It would be useful for most of the IDE users to learn more commands	Closed-ended, five point Likert-Scale with the following answer possibilities: “Strongly agree”, “Agree”, “Neutral”, “Disagree”, or “Strongly disagree”	×	1
10	I think that using an IDE command recommender system would be a good idea	Closed-ended, five point Likert-Scale with the following answer possibilities: “Strongly agree”, “Agree”, “Neutral”, “Disagree”, or “Strongly disagree”	×	1
11, 13, 15, 17, 19, 21, 23, 25, 27	Did you use such a command before (in any IDE)?	Closed-ended, with the following single answer possibilities: “Yes” or “No”	×	1
12, 14, 16, 18, 20, 22, 24, 26, 28	How familiar are you with the command?	Closed-ended, with the following single answer possibilities: “I have never heard of it”, “I have some idea what it is”, “I have a clear idea what it is”, or “I can explain what it is”	×	1
29	From the list of commands presented before, select one that you do not know yet. (If you know all of the presented commands select the one you find the most interesting.) Mark the one you selected	Closed-ended, with the following single answer possibilities: “Breakpoint Properties”, “Navigate Back”, “Next Editor”, “Open Resource”, “Organize Imports”, “Rename Element”, “Step Over”, “Step Return”, “Surround with Try-Catch”	×	2, 3, 4
30	I would find the command easy to use.	Closed-ended, five point Likert-Scale with the following answer possibilities: “Strongly agree”, “Agree”, “Neutral”, “Disagree”, or “Strongly disagree”	×	2
31	I think that I would need some additional support to actually learn how to use the command.	Closed-ended, five point Likert-Scale with the following answer possibilities: “Strongly agree”, “Agree”, “Neutral”, “Disagree”, or “Strongly disagree”	×	3
32	I would imagine that most people would learn to use this command very quickly.	Closed-ended, five point Likert-Scale with the following answer possibilities: “Strongly agree”, “Agree”, “Neutral”, “Disagree”, or “Strongly disagree”	×	2
33	I find the command presentation unnecessarily complex.	Closed-ended, five point Likert-Scale with the following answer possibilities: “Strongly agree”, “Agree”, “Neutral”, “Disagree”, or “Strongly disagree”	×	3
34	I would find the command useful in my job.	Closed-ended, five point Likert-Scale with the following answer possibilities: “Strongly agree”, “Agree”, “Neutral”, “Disagree”, or “Strongly disagree”	×	2
35	I intend to use the command in the future.	Closed-ended, five point Likert-Scale with the following answer possibilities: “Strongly agree”, “Agree”, “Neutral”, “Disagree”, or “Strongly disagree”	×	2

(*continued on next page*)

Table A3 (*continued*)

No.	Question	Question type	Required	GQM-Question
36	Mark all the parts of the command presentation that you find **necessary** for evaluating the quality of the command recommendation	Closed-ended, with the following multiple answer possibilities: "Command name and description (Sec I)", "Explanation of recommendation (Sec II)", "Usage example (Sec III)"	×	4
37	Mark all the parts of the command presentation that you find **useful** for evaluating the quality of the command recommendation	Closed-ended, with the following multiple answer possibilities: "Command name and description (Sec I)", "Explanation of recommendation (Sec II)", "Usage example (Sec III)"	×	4
38	Do you think that some information is missing in the command presentation?	Closed-ended, with the following single answer possibilities: "Yes" or "No"	×	3
39	If yes, which?	Open-ended		3

Appendix B. Potential correlations

Table B4
Potential correlations between categories of users and their agreement with the statements. The values represent (absolute) numbers of participants. SA = Strongly Agree, A = Agree, N = Neutral, D = Disagree, SD = Strongly Disagree.

Category	It would be useful for me to learn more commands available in my IDE.					I find the command presentation unnecessarily complex.				
	SA	A	N	D	SD	SA	A	N	D	SD
Age										
18–24	3	1	0	0	0	0	0	0	3	1
25–34	5	11	1	1	0	0	1	7	10	0
35–44	2	6	0	0	0	1	0	3	4	0
45–54	2	0	1	1	1	1	2	2	0	0
55–64	0	0	1	0	0	0	0	1	0	0
Experience										
< 2	3	0	0	1	0	0	0	0	3	1
2–4	2	1	0	0	0	0	0	1	2	0
5–9	3	6	1	0	0	0	1	3	6	0
10–19	2	10	0	1	1	1	1	7	5	0
20–30	2	0	0	0	0	1	1	0	0	0
> 30	0	1	2	0	0	0	0	2	1	0
I would find the command easy to use.										
SA						1	0	2	4	0
A						0	1	10	12	1
N						1	1	1	0	0
D						0	1	0	0	0
SD						0	0	0	1	0
I would find the command useful in my job.										
SA	3	3	1	0	0					
A	9	10	1	1	0					
N	0	5	0	0	0					
D	0	0	1	1	0					
SD	0	0	0	0	1					

References

[1] G. Fischer, User modeling in human–computer interaction, in: User Modeling and User-Adapted Interaction, Springer, 2001, pp. 65–86.
[2] E. Murphy-Hill, Continuous social screencasting to facilitate software tool discovery, Proceedings of International Conference on Software Engineering. (2012).
[3] D. Campbell, M. Miller, Designing refactoring tools for developers, in: Proceedings of Workshop on Refactoring Tools, 2008.
[4] T. Grossman, G. Fitzmaurice, R. Attar, A survey of software learnability: Metrics, methodologies and guidelines, in: Proceedings of SIGCHI Conference on Human Factors in Computing Systems, 2009.
[5] F. Ricci, Recommender systems: models and techniques, Encyclopedia of Social Network Analysis and Mining, Springer, 2014.
[6] P. Resnick, H.R. Varian, Recommender systems, in: Proceedings of Communications of the ACM, 40, 1997.
[7] F. Ricci, L. Rokach, B. Shapira, Recommender systems: introduction and challenges, Recommender Systems Handbook, Springer, 2015.
[8] I.C. Society, P. Bourque, R.E. Fairley, Guide to the Software Engineering Body of Knowledge, 3rd ed., IEEE Computer Society Press, 2014.
[9] B. Walraet, A Discipline of Software Engineering, Elsevier Science, 2014.
[10] W. Scacchi, Understanding software productivity: towards a Knowledge-based approach, Int. J. Softw. Eng. Knowl. Eng. 1 (1991) 293–321.
[11] I. Sommerville, Software Engineering, International Computer Science Series, Pearson, 2011.
[12] A. Ahmed, Software Project Management: A Process-Driven Approach, An Auerbach book, Taylor & Francis, 2011.
[13] E. Murphy-Hill, G.C. Murphy, Recommendation delivery, Recommendation Systems in Software Engineering, Springer, 2014.
[14] E. Murphy-Hill, R. Jiresal, G.C. Murphy, Improving software developers' fluency by recommending development environment commands, in: Proceedings of ACM SIGSOFT International Symposium on the Foundations of Software Engineering, 2012.
[15] S. Zolaktaf, G.C. Murphy, What to learn next: recommending commands in a feature-rich environment, in: Proceedings of International Conference on Machine Learning and Applications, 2015.
[16] M. Gasparic, T. Gurbanov, F. Ricci, Context-Aware Integrated Development Environment Command Recommender Systems, in: IEEE/ACM International Conference on Automated Software Engineering, 2017.

[17] N. Tintarev, J. Masthoff, Explaining recommendations: design and evaluation, Recommender Systems Handbook, Springer, 2015.

[18] A.R. Hevner, S.T. March, J. Park, S. Ram, Design science in information systems research, MIS Q. 28 (2004) 75–105.

[19] M. Gasparic, A. Janes, F. Ricci, M. Zanellati, GUI design for IDE command recommendations, in: Proceedings of International Conference on Intelligent User Interfaces, 2017.

[20] H.A. Simon, The Sciences of the Artificial, 3rd ed., MIT Press, 1996.

[21] S.T. March, G.F. Smith, Design and natural science research on information technology, Decis. Support Syst. 15 (4) (1995) 251–266.

[22] P. Offermann, S. Blom, M. Schönherr, U. Bub, Artifact types in information systems design science – a literature review, in: Proceedings of International Conference on Global Perspectives on Design Science Research, 2010.

[23] M. Gasparic, A. Janes, F. Ricci, Development tools usage inside out, in: Proceedings of International Conference on XP, 2016.

[24] V. Basili, A. Trendowicz, M. Kowalczyk, J. Heidrich, C. Seaman, J. Munch, D. Rombach, Aligning Organizations Through Measurement: The GQM+Strategies Approach, Springer Publishing Company, Incorporated, 2014.

[25] V.R. Basili, D.M. Weiss, A methodology for collecting valid software engineering data, IEEE Trans. Softw. Eng. 10 (6) (1984) 728–738.

[26] J. Anderson-Meger, Why Do I Need Research and Theory?: A Guide for Social Workers, Taylor & Francis, 2016.

[27] M. Gasparic, G.C. Murphy, F. Ricci, A context model for IDE-based recommendation systems, J. Syst. Softw. 128 (2017) 200–219.

[28] J. Nielsen, Paper versus computer implementations as mockup scenarios for heuristic evaluation, in: Proceedings of Interational Conference on Human–Computer Interaction, 1990.

[29] R. Sefelin, M. Tscheligi, V. Giller, Paper prototyping – what is it good for? A comparison of paper- and computer-based low-fidelity prototyping, in: Proceedings of Extended Abstracts on Human Factors in Computing Systems, 2003.

[30] J. Brooke, SUS-A quick and dirty usability scale, Usability evaluation in industry, Taylor and Francis, 1996, pp. 189–194.

[31] V. Venkatesh, M.G. Morris, G.B. Davis, F.D. Davis, User acceptance of information technology: toward a unified view, MIS Q. 27 (2003) 425–478.

[32] S. Merriam, Qualitative Research: A Guide to Design and Implementation, Jossey-Bass higher and adult education series, John Wiley & Sons, 2009.

[33] A. Bogner, B. Littig, W. Menz, Das Experteninterview: Theorie, Methode, Anwendung, Leske & Budrich, 2002.

[34] U. Flick, An Introduction to Qualitative Research, SAGE Publications, 2009.

[35] A. Strauss, J. Corbin, Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, SAGE Publications, 1998.

[36] A. Bryant, K. Charmaz, The SAGE Handbook of Grounded Theory: Paperback Edition, SAGE Publications, 2010.

[37] M. Zanker, M. Schoberegger, An empirical study on the persuasiveness of fact-based explanations for recommender systems, in: Proceedings of IntRS Workshop, 2014.

[38] F. Gedikli, D. Jannach, M. Ge, How should I explain? A comparison of different explanation types for recommender systems, Int. J. Hum. Comput. Stud. 72 (4) (2014) 367–382.

[39] A. Jameson, B. Berendt, S. Gabrielli, F. Cena, C. Gena, F. Vernero, K. Reinecke, Choice architecture for human-computer interaction, Foundations and Trends in Human-Computer Interaction, volume 7, Now Publishers Inc., 2014, pp. 1–235.

[40] R. Molich, J. Nielsen, Improving a human–computer dialogue, Commun. ACM 33 (1990) 338–348.

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

[42] S. Stuckemann, Vignelli – Automated Design Guidance for Developers, Imperial College London, Department of Computing, 2015.

[43] F. Linton, H.-P. Schaefer, Recommender systems for learning: building user and expert models through long-term observation of application use, in: User Modeling and User-Adapted Interaction, 10, Springer, 2000, pp. 181–208.

[44] J. Matejka, W. Li, T. Grossman, G. Fitzmaurice, CommunityCommands: command recommendations for software applications, in: Proceedings of ACM Symposium on User Interface Software and Technology, 2009.

[45] M. Wiebe, D.Y. Geiskkovitch, A. Bunt, Exploring user attitudes towards different approaches to command recommendation in feature-rich software, in: Proceedings of International Conference on Intelligent User Interfaces, 2016.

[46] M. Gasparic, GUI Evaluation Data for an IDE Command Recommender System, 2017, https://doi.org/10.5281/zenodo.581390.

[47] J.L. Herlocker, J.A. Konstan, J. Riedl, Explaining collaborative filtering recommendations, in: Proceedings of ACM Conference on Computer Supported Cooperative Work, 2000.

[48] R. Sinha, K. Swearingen, The role of transparency in recommender systems, in: Proceedings of Extended Abstracts on Human Factors in Computing Systems, 2002.

[49] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, Experimentation in Software Engineering, Springer, 2012.

[50] R.K. Yin, Case Study research, Design and Methods, Applied Social research Methods Series, 3rd ed., SAGE Publications, 2009.

[51] S.E. Dreyfus, H.L. Dreyfus, A Five-Stage Model of the Mental Activities Involved in Directed Skill Acquisition, California Univ Berkeley Operations Research Center, 1980.

[52] M.A.A. Khan, V. Dziubak, A. Bunt, Exploring personalized command recommendations based on information found in web documentation, in: Proceedings of International Conference on Intelligent User Interfaces, 2015.

[53] L. Ponzanelli, G. Bavota, A. Mocci, M.D. Penta, R. Oliveto, M. Hasan, B. Russo, S. Haiduc, M. Lanza, Too long; didn't watch!: extracting relevant fragments from software development video tutorials, in: Proceedings of International Conference on Software Engineering, 2016.

[54] C.D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, 2008.

[55] M. Gasparic, F. Ricci, Should context-aware ide command recommendations always be presented in-context or not? in: Proceedings of AWARE workshop, 2017.

[56] W. Li, J. Matejka, T. Grossman, J.A. Konstan, G. Fitzmaurice, Design and evaluation of a command recommendation system for software applications, ACM Trans. Comput.-Hum. Interaction 18 (2011) 6:1–6:35.