Received December 27, 2019, accepted January 10, 2020, date of publication January 20, 2020, date of current version January 30, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2967840

IDE Interaction Support With Command Recommender Systems

MARKO GASPARIC[1] AND FRANCESCO RICCI[2]
[1]Faculty of Electrical Engineering and Computer Science, University of Maribor, 2000 Maribor, Slovenia
[2]Faculty of Computer Science, Free University of Bozen–Bolzano, 39100 Bolzano, Italy

Corresponding author: Marko Gasparic (marko.gasparic@um.si)

This work was supported in part by the Slovenian Research Agency under Grant P2-0057.

ABSTRACT Software developers' knowledge of integrated development environment (IDE) directly impacts on their productivity. IDE command recommender systems aim at identifying and convincingly presenting to software developers functionality that can help them to accomplish their daily tasks, without overloading them with well known or useless information. Command recommendation requires the estimation of both the utility of commands and the acceptance of the user for new command recommendations. In this paper, we focus on how and when such recommendations should be presented. We performed a long-term user study and our results show that IDE command recommendation must be presented with adequate descriptions of the commands and good usage examples. It seems that a higher frequency of recommendation notifications could be useful, but it should not be too intrusive, especially while developers are focusing on more demanding tasks. To improve recommendation acceptance rate, researchers should also focus on context-aware algorithms and tailor command recommendation timing.

INDEX TERMS Command, delivery, integrated development environment, presentation, recommender system, software development.

I. INTRODUCTION

Software developers often perform their tasks in integrated development environments (IDEs), which integrate multiple tools, such as, advanced source code editors, testing tools, automatic compilers, and debuggers. It is well known that development tools affect software development efficiency and quality [1], by speeding up repetitive tasks and allowing programmers to focus on the more critical aspects of the process [2].

IDE commands are shortcuts and menu buttons that execute different functions, such as Save, Run, Open Resource, and many others. To improve command knowledge, IDE vendors and researchers devised different types of recommender systems (RSs). In software engineering, RSs are defined as software applications that provide information estimated to be valuable for a software engineering task in a given context [3]. For example, the provided information can be the suggestion to use a (not yet used) command.

In fact, very often, in high-functionality applications, such as IDEs, the simple lack of adequate tool knowledge is the main reason for potentially useful functionality not being used [4]. Additionally, as already mentioned, the tool knowledge directly impacts on the productivity gain that the tool is supposed to bring [1]. Moreover, since IDEs are complex systems serving the needs of a large and diverse user population, it is not required that a generic user masters the entire provided functionality. For example, an average Eclipse user uses 42 different commands, out of more than 1100 that are available [5], which is probably not enough for an effective IDE use, but clearly not all these thousand commands are needed by all.

Therefore, *what* will be recommended is an important question to answer when building an IDE command RS (ICRS). Nonetheless, an additional aspect to consider, after having identified useful command recommendations for a specific user, is *how* and *when* these recommendations should be presented to the user. This aspect is often neglected and it is instead among the key analysis conducted in this paper.

At the moment, two general types of help systems, for proactively improving the knowledge of available functionality, can be found in IDEs. We call them *tips* and *quick assists*. The first one is often implemented as a "Tip of the Day" window, at the launch of the application. This is not limited

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

to IDEs, actually, it is used in diverse high-functionality applications. However, users often get annoyed by such recommendations, especially when they are not able to identify any really relevant information or when the recommendations refer to already known commands [6].

In the second type of help solutions, suggestions are introduced with a light-bulb icon that lets the programmer to open a drop-down list of recommendations that help to fix the issue. This approach is frequently used to support software developers when they are editing source code that does not compile or has some other errors. While items in these recommendation lists are not limited to IDE commands, they often include some. This type of functionality is called Quick Fix and Quick Assist in Eclipse, Quick Action in Visual Studio, and Intention Action Suggestions in IntelliJ IDEA. It could have the side effect to inform the user about new editing commands.

ICRS research community has designed, implemented, and tested several algorithms (see [7]–[12]), graphical user interfaces (GUIs) (see [13]–[15]), and even complete systems for recommending IDE commands in different settings (see [14]–[16]). However, these RSs have not been integrated in popular IDEs, yet; arguably because the accuracy and acceptance of the recommendations are still too low to enable practical usage.

There have been several attempts to apply techniques developed for RSs in other domains, but they failed to effectively tackle the substantially novel issues of IDE command recommendation problem specificity. We argue that algorithms, guidelines for user interaction, or results of evaluations, which have been developed in other fields, cannot be directly applied in ICRS.

The overall goal of our research is to advance the state of the art in the design of effective ICRS. This is a general objective that requires addressing some specific research questions. First, it is important to understand whether the quality of the algorithmically generated recommendations should be improved and how this can be achieved (research question RQ1, in the reminder). Then, we would like to know how ICRS users, in general, learn new commands and if they are aware of how they have learned them (RQ2). Then, we are interested in understanding why users reject certain recommendations (RQ3). And finally, because ICRS user interfaces have not been produced and analysed extensively, we want to explore this subject further: we want to know which parts of the GUI are useful (RQ4) and when ICRS users interact with the system (RQ5).

In this paper, we first analyse the IDE command recommendation problem (Sec. II) and relevant related work (Sec. III). We continue by presenting the protocol, validity threats, and results of a study that lasted 40 weeks; during which we observed 697,485 executions of 282 distinct IDE commands and delivered 508 recommendations, out of which 93 were accepted (Sec. IV). Finally, we provide a discussion of not yet published results and potential improvements in ICRS GUIs (Sec. V), before we conclude (Sec. VI).

We note that some of the results of this study were already published at the International Conference on Software Engineering (ICSE), which is the premier software engineering conference. They are presented in [16], which is focused on IDE command knowledge improvement. Hence, there is a partial overlap between [16] and this paper, related to RQ1. We include here a summary of relevant results shown in [16] because we believe that RQ1 is crucial for understanding the context in which the other four questions were answered.

II. PROBLEM STATEMENT

ICRSs aim at identifying and convincingly presenting to IDE users the right functionality that can help them to accomplish their daily tasks, without proposing well known or useless information. The number of recommended commands that a user can learn from an ICRS is not limited. In this sense, accepting one recommendation does not exclude the acceptance of another, which is typical in other RSs where the user is supposed to choose from a set of alternative items (e.g. a movie to watch). For example, the acceptance of a recommended Navigate Back command does not affect the relevance of another command recommendation, such as Debug or Organize Imports, which are also worth learning.

In IDE command recommendation, the cost of making a wrong decision is low: the rejection of a good recommendation preserves status quo in the development process, while the acceptance, i.e., execution, of a bad recommendation sometimes results in an undesired modification of the artefacts under development, which fortunately can be often reverted with minimal effort. If command recommendations are presented adequately, the time needed to take a decision can be short. In our study, as we will present also in Sec. IV, the median time spent for reading command recommendations was less than 11 seconds.

On the other hand, fully understanding complex commands may require a considerable cognitive effort, which may be higher than what the developer is willing to invest. In case of a bad recommendation, this effort may even exceed the utility of the recommended command. And, on top of this, the acceptance of recommendations requires also recipients' awareness and acknowledgement of a deficit in IDE usage, which may be unpleasant for certain developers [13].

As our results show, it is necessary, and likely sufficient, to provide a good command description and usage example for every recommendation. Then, if the recipients do not forget about the command, do not consider it useless, or are not already using some other commands for the same purpose, the command will probably be executed in the future.

Furthermore, popular IDEs typically include a few hundred commands and command RSs have to rely exclusively on implicit feedback (i.e., user actions) to infer the users' opinions about the commands. RSs in other domains, such as book or movie, are different, since they operate on much larger datasets with thousands or even millions of items in the catalogue, and they still collect explicit feedback, such as star-ratings or thumbs up/down.[1] It is unlikely that minor adaptations to mainstream algorithms may suffice; in fact, existing research indicates that this is not the case [8], [9].

Once the command recommendations are identified, they also have to be presented with a proper GUI that allows the recipients to decide whether to accept them or not [17]. The two major issues related to a RS's user interface are: what will be visualised and when. One could think that the holy grail of ICRSs research is being able to recommend a command that automates an action when the developer is trying to perform the same task manually or in some other suboptimal way. However, this could as well be a myth, because an instant execution of a recommended (unknown) command would require currently unattainable levels of the system's accuracy and users' trust. For example, best performing neural networks achieve 64% accuracy in predicting what command a user will execute next [18]. Besides, if the recipient decides to consider such a recommendation, she has to interrupt her work, which is rarely desired.

In practice, two main strategies for offering command recommendations can be implemented, namely: *global* suggestions, where the system provides general recommendations, not necessarily justified by the precise current state of the user-system interaction or user activity; and *opportunistic* suggestions, where the system mainly takes into account what the user is doing at a particular moment and it provides recommendations that are highly relevant in that context [19]. A typical example of *global* suggestions, already implemented in many IDEs, are *tips* visualised at the launch of the application. Conversely, a typical example of *opportunistic* suggestions are commands in a drop-down list of *quick assist* recommendations that can fix a source code error. If compared with the help systems implemented by industry, regardless of the adopted recommendation strategy, ICRSs suggested by researchers should aim for a higher acceptance rate than *tips* and should be able to recommend a wider spectrum of commands than *quick assists*.

We used CoRe ICRS [16] in the study reported in this paper, which is focused on *global* suggestions. Even though we have not tested an *opportunistic* system that would genuinely adapt recommendations to a specific context, the right timing for recommendations must be determined and from our results it seems clear that recommendation recipients prefer to interact with the system when they are not working. Interestingly, this was the case also in the computer-aided design domain, where command recommendations complied with the *opportunistic* strategy [19].

Finally, the recommendation presentation content plays an important role in the usage of any RS. The GUI can affect user's trust and loyalty, it can change the user decision to interact with the RS, and it can also influence user's final decision about the recommendation [20]. Moreover, based on our experience, different GUI elements require different levels of effort to be produced. For instance, videos of usage examples are much more costly, in terms of required effort, than command description. Thus, it is necessary to conduct more research to better understand which information should be included in the command recommendation presentation. As already mentioned, we provide some new findings below.

It is also important to select a proper approach to notify the user about the existence of recommendations. We assume that every ICRS should be (somewhat) proactive, i.e., it will not generate and present recommendations only at an explicit user request. Consequently, there is a trade-off between the level of potential distraction and decreased value due to a delay [17]. A common balancing approach is the implementation of a *negotiated interruption* [21], which informs the user about available recommendations, but does not force any specific action. Obviously, there are alternatives, such as: present a blocking dialogue with a recommendation, present a dialogue with a notification and link, force the user to take a tutorial, change focus, etc.

Combined with a selected notification approach, there are many possibilities on when should a command be visualised or when should ICRS users be informed about the existence of new recommendations, such as: notify about the recommendation when it is most useful, when the developer is stuck, when there is a high chance of interaction with the ICRS, on Monday morning, on Friday afternoon, when IDE starts, etc. Up to date, the systems are simply showing recommendations when they are generated or at regular intervals, which is simple, but tends to cause unjustified interruptions and may indirectly reduce the acceptance rate.

In our study, we generated recommendations for six consecutive weeks, and on every Thursday, with a notification dialogue and a change of color of the menu icon, we informed the students that new recommendations are available. Our results show that a higher frequency of recommendation notifications could be rewarding.

III. RELATED WORK

ICRSs typically consist of three components, namely: a data collection module, which automatically accumulates required data, such as executed commands, time, project artifacts' metadata, etc.; an algorithm that generates command recommendations, using the collected data; and a graphical user interface, which is used for presenting recommendations to the recipients. Most of the previous research focused on the algorithmic component, while the importance of the command presentation and the quality of the collected data have been largely neglected.

A. IDE COMMAND RECOMMENDER SYSTEMS

We are aware of three fully implemented ICRSs that have been developed in the past. Spyglass [15] observes and analyzes user behavior to check whether certain steps could be replaced by IDE commands. The input data is the IDE command execution history.

[1] Implicit feedback is getting more important also in these domains but they are usually combined with reviews and ratings.

Vignelli [14] detects potential design flaws in the source code, such as bad smells, and guides the developer through the refactoring process. If an action in the refactoring process can be automated, Vignelli recommends suitable commands. The input data is the source code of the project.

CoRe [16] observes development contexts in which developers work and execute commands. It suggests commands that could be relevant for the recommendation recipient's work, but are never used. Additionally, CoRe can use alternative algorithms, such as those based on command popularity or collaborative filtering.

The input data for CoRe ICRS consists of IDE command execution histories belonging to different users and contextual information according to the context model suggested by Gasparic *et al.* [22]. The model consists of 13 factors that describe the situations in which developers interact with the IDE, including: development activities, characteristics of the source code under development, and state of the IDE instance, described with visible and active GUI elements. The inclusion of the model in CNTX algorithm [8], which is the main recommendation algorithm in CoRe, is supported by the model evaluation results, which show that developers execute different commands in different contexts [22].

Regarding the ICRSs utility, Spyglass evaluation showed that an ICRS can make developers aware of available tools to a similar degree to a tutorial, but with less effort [15]. Murphy-Hill et al. showed that it is feasible to automatically recommend useful IDE commands to software developers [9]. And CoRe evaluation showed that IDE command recommendations successfully promote the exploration of the IDE functionality, no matter whether they are based on complex or simple algorithms, while the actual acceptance rate is low in a real-life setting [16].

B. ALGORITHMS FOR IDE COMMAND RECOMMENDER SYSTEMS

There is a wide spectrum of command recommendation algorithms. Murphy-Hill *et al.* [9] used six algorithms based on command popularity and collaborative filtering, by taking into account command executions histories. Zolaktaf and Murphy [12] suggested CoDis, which is based on command discovery patterns and co-occurrence of executions in sessions. Silva *et al.* [11] performed static source code analysis to identify and rank opportunities for refactoring command executions. Gasparic *et al.* [8] suggested CNTX, an algorithm that observes which commands are executed in which development contexts and trains a regression model to generate recommendations. Damevski *et al.* [7] used IDE-interaction data and applied topic modelling to predict future behaviour of developers, which was ultimately used for command recommendation generation. Schmidmaier *et al.* [10] applied user modelling on tasks that were extracted from the IDE-interaction data, to recommend commands associated with a task that has the maximum payoff, according to a multi-armed bandit problem optimization.

C. USER INTERFACES FOR COMMAND RECOMMENDER SYSTEMS

When it comes to IDE command recommendation presentation, a few papers focus on this issue. Gasparic *et al.* [13] designed and evaluated an ICRS GUI mock-up, which was updated, based on the user evaluation, and implemented for the purposes of another study reported by Gasparic *et al.* [16] and also in the study reported in this paper.[2] Schmidmaier *et al.* [10] presented a set of ideas on possible ICRS GUIs, but did not provide a comprehensive description that could lead to implementation. Gasparic and Ricci [23] published a discussion on whether context-aware recommendations should always be provided ''in context'', to which the answer is: ''Not necessarily''.

Essentially, we are left with the GUIs implemented in previously mentioned complete ICRSs, however, the assessment of the GUI or detailed analysis of the user interaction are not provided in the related evaluations.

Despite the fact that Spyglass [15] presents *opportunistic* recommendations and CoRe [16] presents *global* recommendations, these two GUIs are similar. They both change the color of the menu icon and present a pop-up to inform the developer about new recommendations. They present recommendations in a list and when a command is selected, they open a dialogue with the command explanation including a recommendation rationale, shortcut, and a brief description of the automated function. They differ in that CoRe includes usage examples and Spyglass includes menu navigation path.

The GUI in Vignelli [14] shows command recommendations in the text describing the required refactoring process. It notifies the developer about recommendations by highlighting the parts of the source code that should be refactored.

We would also like to mention some other GUIs, which have been designed for command RSs in other domains, but they may be related to ICRS applications. In OWL [24], a list of Microsoft Word command recommendations is displayed in a view that can be opened by clicking on a button in the application toolbar; clicking on the name of the command in the list results in a presentation of a short description and the RS's score, i.e. predicted utility for the user. In CommunityCommands [25], AutoCAD command recommendations are presented in a palette, with a visible name and relevance to the user's current workflow, and a short description of the command in a tooltip. In QFRecs [26], two different user interfaces for presenting GIMP command recommendations are implemented: in the ''combined'' interface, in the application menu, recommendations for familiar commands are highlighted with blue and for unfamiliar with pink color, and in ''separated'' interface, unfamiliar commands are presented in a palette, with full menu paths presented in the tooltips. And finally, Wiebe *et al.* [27] suggested two command RSs for

[2]We provide a detailed description of the updated GUI in the next section.

GIMP, both presenting recommendation lists, where clicking on a command results in the visualization of the command description and system's confidence of the appropriateness of the recommendation, together with the percentage of all and similar users using the command in the "social" RS and a list of tasks related to this command in the "task-based" RS.

D. RECOMMENDER SYSTEMS IN OTHER DOMAINS

In the conclusion of this section, we would like to stress that RSs for software engineering and command RSs represent two relatively new domains of application, which evolved from broader RSs research. More traditional RSs are defined as personalized information search and filtering tools that suggest useful items [28]. People encounter them daily, when they are shopping on Amazon, selecting a movie on Netflix, or building a network on LinkedIn.

The main differences between traditional RSs and ICRSs are that ICRSs do not rely on explicit feedback, such as ratings; the catalogue size of recommended items is much smaller for ICRSs, but item complexity is typically higher; and traditional RSs usually suggest a list of items, so that the users may select (choose) one of them, mainly according to their taste, and be satisfied with the selection. On the other hand, ICRSs aim to convince the users that their interaction with an IDE has to change, since all the recommended items are worth being used during their work.

IV. LONG-TERM USER STUDY

We evaluated CoRe ICRS in a real-life setting, with first year bachelor students. For 40 weeks, which is the entire school year, we collected IDE-interaction events, such as, command executions, editing actions, opening and closing of different IDE views and other IDE GUI elements, etc. Altogether, we detected 148 different user identifiers.

The time window of interest is from the beginning of the recommendation generation, which started in week 26, until the end of the study, i.e., the last 14 weeks. There were 19 users, out of previously mentioned 148, who received command recommendations.

The research questions addressed in this paper, which we already mentioned in the Introduction, are:

- RQ1: What is the perceived recommendation quality?
- RQ2: How users learn new commands and how much they are aware of how they learn them?
- RQ3: What are the reasons for rejections of recommendations?
- RQ4: Which GUI parts are important and useful?
- RQ5: When are users interacting with the ICRS?

A. RESEARCH METHOD

To evaluate the quality of CoRe ICRS, assess its effect on user behaviour, and study user interaction with the ICRS, we invited the students of the Introduction to Programming and Advanced Programming courses, at the Free University of Bozen–Bolzano, to participate to our study. The majority of the study participants installed the monitoring tools, listed below, in October 2016; others were invited to join the study at the beginning of March 2017, when the Advanced Programming course began. Data collection started on October 12th 2016 and stopped on July 19th 2017. The main milestones are visualized in Fig. 1.

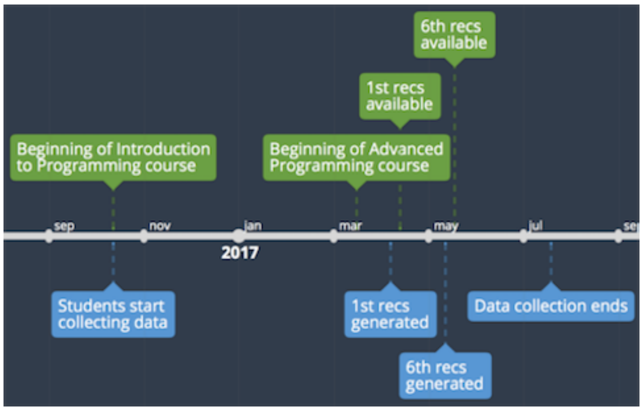

6th recs available
1st recs available
Beginning of Introduction to Programming course
Beginning of Advanced Programming course
sep
nov
jan
mar
may
jul
sep
2017
Students start collecting data
1st recs generated
Data collection ends
6th recs generated

FIGURE 1. Study milestones on a timeline.

To participate in the study, the students were requested to install a custom version of Eclipse IDE, which already included the necessary monitoring tools, namely, Eclipse UDC[3] and a special version of Mylyn, suggested by Kersten and Murphy [29], which we call Lema.[4] When Eclipse is running, these two monitoring tools automatically log interaction events and relevant context information, which is periodically uploaded to our database.

On April 13th 2017, we asked the students to install the GUI plugin for recommendations visualisation. During the next 6 weeks, on every Thursday morning, each of 19 volunteers who installed the GUI plugin received top-5 recommendations. Thus, every volunteer received 30 command recommendations, in total. Recommendations were generated by three different algorithms:

- Most Popular[5]: baseline algorithm, often used for comparison of RSs also in other domains;
- CoDis[6]: the algorithm that performed best in the offline study reported by Zolaktaf and Murphy [12]; and
- CNTX[7]: a context-aware algorithm, which was developed by us and it is described in [8].

Each user received recommendations from each algorithm for two consecutive weeks. The sequence of the algorithms was random, on the individual level, but we assured that there were only small differences between the overall numbers of participants who received recommendations in a certain

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

[4]https://sites.google.com/site/mgasparic/eclipse

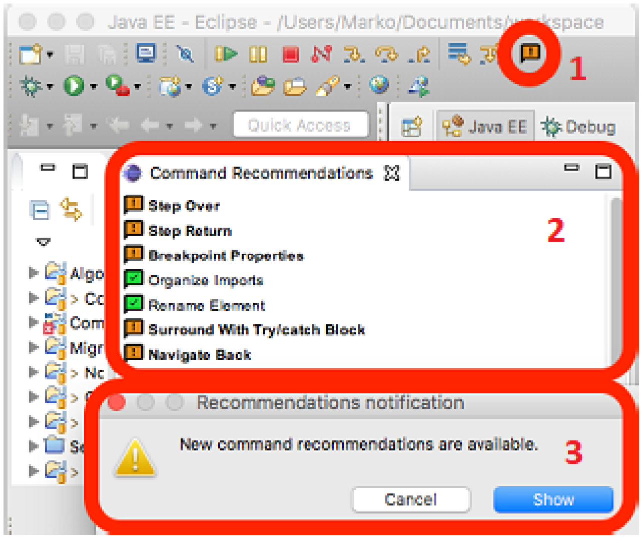
[5]We used the implementation available at https://github.com/ubc-cs-spl.

[6]We used the implementation provided by the authors.

[7]We used the implementation available at https://gitlab.inf.unibz.it/tural-gurbanov/ide_rs.

chronological order.[8] Users had no information on how recommendations were generated.

The first set of recommendations was available on April 13th 2017, which is 26 weeks after the beginning of the data collection process, when also the Eclipse plugin for visualising recommendations was published and the students were invited to install it. The recommendation algorithms require to be trained, hence, we used as training set all the data collected between October 12th 2016 and Friday evening before the Thursday on which recommendations were available. The delay of six days was necessary because the execution of training and command recommendation generation required some time and we also had to manually prepare the descriptions of the recommendations, which was very time-consuming, particularly in the first two weeks.



New command recommendations are available.

FIGURE 2. ICRS toolbar icon (1), Recommendation List view (2), and Notification dialogue (3).

When new recommendations were available, the GUI showed a dialogue in Eclipse, as shown in Fig. 2. If the Recommendation List view was closed, the user could open it by selecting the "Show" button in the dialogue or by clicking on the ICRS toolbar icon. Next to the command, in the Recommendation List view, an icon also showed whether the recommendation was already opened. When a user clicked on a recommendation in the list, the selected command was presented in the Command Recommendation view, which included the command name, shortcut, description, usage example, link to a step-by-step video guide published on YouTube, and the explanation of the recommendation rationale, which had to be selected, before it was visualized, as can be seen in Fig. 3.

We continuously monitored the usage of the IDE commands and the interaction with the ICRS. We logged every visualization and every click on the interactive GUI elements, i.e., buttons, video link, and explanation. Beside the 19 volunteers who installed all the required plugins and were able to receive recommendations, we detected 129 other users.

Altogether, during the 40 weeks of data collection, we detected 697,485 executions of 282 distinct commands. An average user identifier is associated with 4,713 executions of 32 distinct commands.[9] If we only look at the sample of 19 volunteers, we see that an average user identifier is associated with 16,529 executions of 80 distinct commands. Hence, it is evident that this sample of users, who installed the GUI, is not representative of the entire population of study participants.

Since we generated 30 recommendations for each of the 19 users, 570 recommendations were supposed to be delivered. But, because we were able to monitor when the content of the Recommendation List views was visible on the screen, we could check that only 525 commands were actually visualized, i.e., at least the names of the recommended commands could be seen in Eclipse IDEs used by different participants. 170 of these commands were recommended by the Most Popular algorithm, 175 by CoDis, and 180 by CNTX.

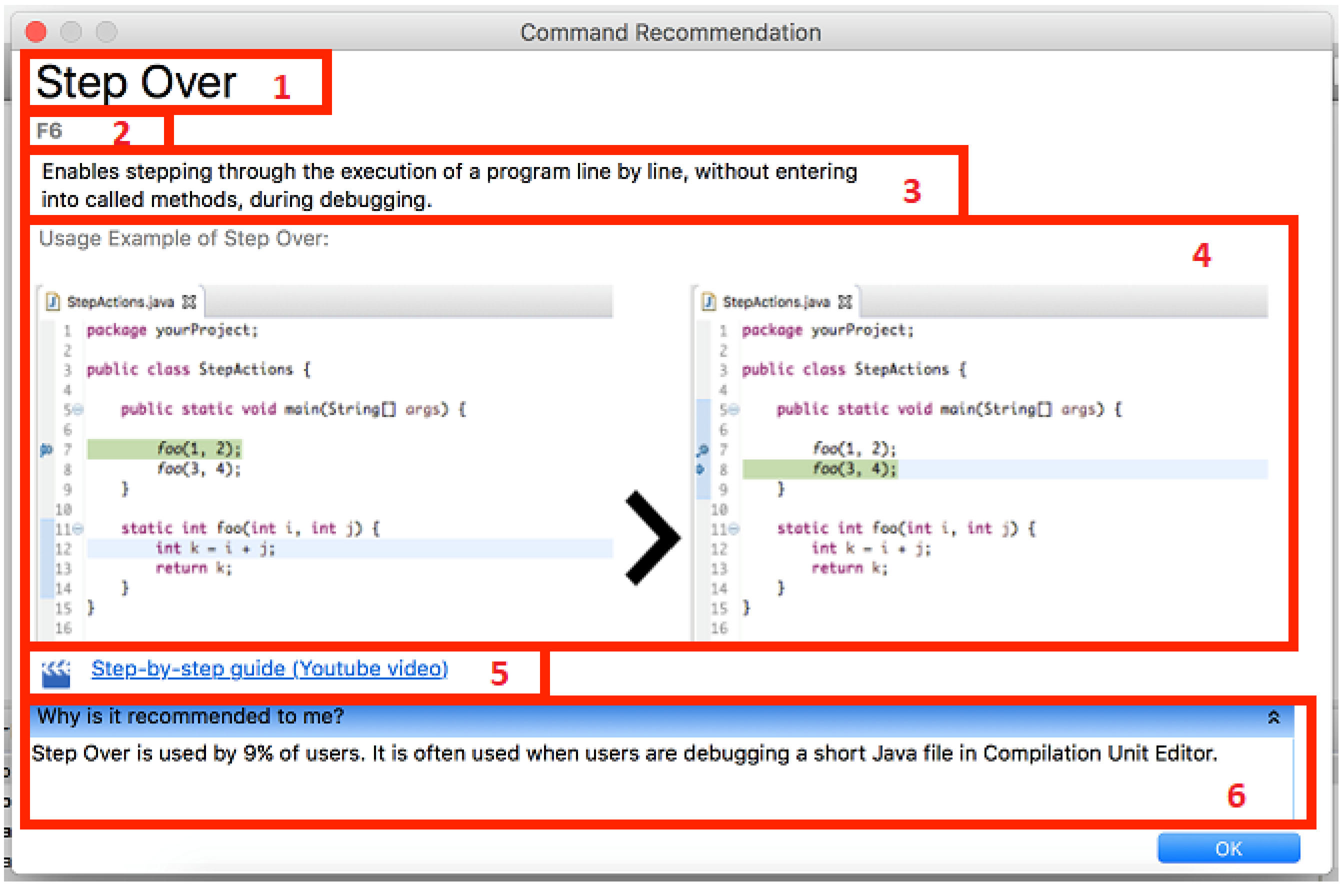
To obtain additional information, we asked the recommendation recipients to answer short questionnaires. The details of the questionnaires are presented in Tab. 1. Remember that the participants did not know that we used different algorithms to generate recommendations, hence, they evaluated the system as a whole.

In the first questionnaire, we asked the participants how familiar are they with the recommended commands. The set of possible answers was based on the taxonomy proposed by Anderson-Meger [30]. The question was delivered via Eclipse dialogue every time the Command Recommendation view (see Fig. 3) was closed, until the participant answered the question related to the specific command. We received 290 answers from 17 different participants. The analysis of this data is out of the scope of this paper, but we mention this questionnaire on the sake of completeness and for better understanding of the full study execution protocol.

In the second questionnaire, the participants were asked four questions. Firstly, we asked them how they learned about the command and we provided a set of possible answers, which is similar to the one used by Murphy-Hill *et al.* [31]. The answers to this question were used to answer RQ2. We also asked the participants to mark on a five-point Likert scale how strongly they agree with three UTAUT [32] statements, however, as for the first questionnaire, the second part

[8]The number of users who received recommendations according to a certain algorithm sequence is quite similar to others. For example, 4 users, which is the size of the three largest groups, received recommendations in the following order: "weeks 1 and 2: Most Popular, weeks 3 and 4: CNTX, weeks 5 and 6: CoDis", and 2 users, which is the size of two smallest groups, received recommendations in the following order: "weeks 1 and 2: CNTX, weeks 3 and 4: Most Popular, weeks 5 and 6: CoDis".

[9]As expected, the distribution of the command executions has a long tail: 11 most active users contributed almost 50% of command executions, 25 most active contributed more than 80%, and 64 most active users contributed more than 99% of command executions. Overall, the most used command is Undo, with 118,520 executions; it is closely followed by the Paste command, with 117,646 executions. On the other side, 20 commands were executed only once.



Step Over

F6

Enables stepping through the execution of a program line by line, without entering into called methods, during debugging.

Usage Example of Step Over:

```
package yourProject;

public class StepActions {

    public static void main(String[] args) {

        foo(1, 2);
        foo(3, 4);
    }

    static int foo(int i, int j) {
        int k = i + j;
        return k;
    }
}
```

```
package yourProject;

public class StepActions {

    public static void main(String[] args) {

        foo(1, 2);
        foo(3, 4);
    }

    static int foo(int i, int j) {
        int k = i + j;
        return k;
    }
}
```

Step-by-step guide (Youtube video)

Why is it recommended to me?

Step Over is used by 9% of users. It is often used when users are debugging a short Java file in Compilation Unit Editor.

FIGURE 3. Recommended command name (1), shortcut (2), description (3), usage example (4), link to the video guide (5), and recommendation explanation (6).

of the second questionnaire is also out of the scope of this study.

The second questionnaire was delivered via Eclipse dialogue after a new command discovery was detected. The command was considered discovered only if it was executed at least five times and was not executed before 7th April. To find newly discovered commands, we analysed the IDE-interaction history logs every week, for 6 weeks, which is the duration of the recommendation generation period, but we started with a delay of two weeks, to give time to the users to adopt the recommendations. We stress that we asked the user to fill the second questionnaire even if the command was never recommended. We detected 103 newly discovered commands, but we only showed one questionnaire at a time and some users eventually stopped answering. Consequently, the number of shown questionnaires is 69 and the number of answered questionnaires is 53. The answers were provided by 12 different participants.

The third questionnaire contained 8 different questions. Firstly, we asked what the participants think about the quality of the recommendations, using three statements suggested by Knijnenburg *et al.* [33] and a five-point Likert scale. The answers to these questions were used to answer to RQ1. We collected 13 answers from 13 different users.

Afterwards, we provided a personalised list of randomly selected commands that were never used, even though they were recommended. For each command, we asked the participants why they did not use it. We note that we limited the size of the list to maximum three commands, in order to achieve higher response rate, by not making the questionnaire too long. Not all participants received three questions in this part, either because they accepted a lot of recommendations or because they did not interact with the CoRe ICRS enough. Altogether, we collected 31 answers from 12 respondents. The answers were used to answer to RQ3.

The last part of the third questionnaire contained four questions that we already used in the mock-up evaluation study [13]. These questions are: which parts of the GUI are necessary and which are useful for evaluating the quality of the command recommendation; and whether some information is missing in the command recommendation presentation and which. For each question we received 13 replies from 13 different users. They are all related to RQ4.

The third questionnaire was available on a web page, to which the link was provided in the IDE. It was delivered one week after the last set of recommendations was available.

TABLE 1. Questionnaire details.

No.	Questions/statements	Possible answers	Timing	Responses	RQ
1	Please tell us how familiar were you with '[NAME]' command, before it was recommended	I never heard of it; I had some idea what it is; I had a clear idea what it is; I could explain what it is.	On Command Recommendation view close event	290 answers 17 respondents	(out of scope)
2	Please tell us from where did you learn about this command	Recommender system recommended it; Noticed someone else using it; Someone (a person) recommended it; Found it on a web page; Found it in Eclipse user interface; Other; Actually, I am not using this command.	After a new command discovery was detected	53 answers 12 respondents	RQ2
	I find the command easy to use	Strongly agree; Agree; Neutral; Disagree; Strongly disagree.		53 answers 12 respondents	(out of scope)
	I find the command useful in my work	Strongly agree; Agree; Neutral; Disagree; Strongly disagree.		53 answers 12 respondents	(out of scope)
	I intend to use the command in the future	Strongly agree; Agree; Neutral; Disagree; Strongly disagree.		53 answers 12 respondents	(out of scope)
3	I liked the commands recommended by the system	Strongly agree; Agree; Neutral; Disagree; Strongly disagree.	Week 33	13 answers 13 respondents	RQ1
	The recommended commands were relevant for my tasks	Strongly agree; Agree; Neutral; Disagree; Strongly disagree.		13 answers 13 respondents	RQ1
	The system recommended too many bad commands	Strongly agree; Agree; Neutral; Disagree; Strongly disagree.		13 answers 13 respondents	RQ1
	Why did you not use this command? (max. 3-times)	This command is not useful for my work; I am using other commands to do the same thing; The command is too hard to use; I forgot about this command; I do not know why I did not use the command; Other; Actually, I was using it.		31 answers 12 respondents	RQ3
	Mark all the parts of the command presentation that you find necessary for evaluating the quality of the command recommendation	Command name and description (Sec I); Explanation of recommendation (Sec II); Usage example (Sec III).		13 answers 13 respondents	RQ4
	Mark all the parts of the command presentation that you find useful for evaluating the quality of the command recommendation	Command name and description (Sec I); Explanation of recommendation (Sec II); Usage example (Sec III).		13 answers 13 respondents	RQ4
	Do you think that some information is missing in the command presentation?	Yes; No.		13 answers 13 respondents	RQ4
	If yes, which?	(open-ended)		0 answers	RQ4

B. VALIDITY THREATS

A number of threats affect the results of our evaluation.

1) CONCLUSION VALIDITY

Conclusion validity is related to the correctness of the conclusions about the relations between the treatment and the outcomes of the study [34]. In our case, the main threat to the conclusion validity is the fact that only 19 students participated. Hence, the ability to reveal correct patterns in the data is decreased. Furthermore, our study is exposed to the threats of the ''reliability of the treatment implementation'' [34]. In fact, we observed that the recommendation acceptance decreases over the weeks. Consequently, the application of the same treatment, which in our case refers to providing recommendations generated by a specific algorithm, is not the same for different persons who received the treatment in different weeks.

In other words, if the set of participants was larger and different from the one we had, e.g., if the participants were less eager to use the ICRS or maybe less skilled than our volunteers, and if the algorithms used had generated different recommendations, leading to either higher or lower acceptance rates, also the interaction with the ICRS could change. Consequently, it is safe to assume that the collected data could be different, just as our conclusions. We partially decreased the potential impact of the two threats by using three different algorithms and randomly selecting the sequence of the treatments for the participants.

The third relevant conclusion validity threat in our study is ''reliability of measures'' [34]. It is possible that when the same phenomenon is measured twice, the outcome is different. That could happen, for example, because of poor question wording or bad instrumentation. To mitigate this threat, we used well tested software to collect the data and standard questionnaires suggested in the literature. When we could, we combined different techniques to measure the same phenomenon and compared the results. For example, we used questionnaires and compared the results with the observations of the user behaviour, which were based on automatically collected data.

2) INTERNAL VALIDITY

Internal validity is related to the correctness of the causal influences that affect the observed variables, without researcher's knowledge [34]. In our case, because the algorithms were used at different times and in a different sequence, it is possible that the history and the maturation of the subjects affected the results; for example, by demoralising some participants with initial bad recommendations. To mitigate this threat, we randomly selected the sequence of the algorithms for each participant. We believe that different effects could average out, however, our sample was rather small and we were not able to control or measure the effect of history and maturation on the results.

The second internal validity threat is related to the selection of the participants, which was based on the voluntary participation. Since volunteers are generally more motivated than the whole population, they are not representative. This was the case also in our study. We noticed that the students who installed the GUI have much better knowledge of IDE functionality than other students, who were also invited to participate, but rejected the invitation.

3) CONSTRUCT VALIDITY

Construct validity is related to the correctness of the results generalisation to the theory behind the study [34]. In our case, a potential threat is ''mono-method bias'' [34], which is similar to ''reliability of measures''. If we used only one measurement method and this method is biased, the results would be wrong. For example, in our study, we observed that the perceived usefulness of recommendation explanation was high for those participants who did not look often to these explanations and low for those participants who opened explanations regularly. This contradictory behaviour of the participants also poses the question whether the perceived usefulness of a certain GUI element can be artificially high in an unreal setting. To decrease this threat, we used automatically collected data to assess the actual effect of the explanation on recommendation acceptance.

4) EXTERNAL VALIDITY

External validity is related to the correctness of the results generalisation to the industrial practice [34]. In our case, the student volunteers who installed the GUI show certain similarities, according to the command knowledge, with the experienced IDE users working in academia and industry, who participated in the case study reported by Gasparic *et al.* [22]. Nevertheless, the setting in which the study took place was not similar to industrial settings, hence, the generalisability of the results is questionable.

C. RESULTS

In this section, we present the results of our study related to the previously introduced research questions.

1) RQ1: WHAT IS THE PERCEIVED RECOMMENDATION QUALITY?

Out of 525 delivered command recommendations, 17 were executed at least once before they appeared in the recipients IDE and we do not take them into account. From the remaining 508 recommendations, 93 were executed at least once and 415 were never executed by the recommendation recipient. Consequently, the recommendation acceptance rate is a bit larger than 18%. This is almost the same as the proportion of the recommended IDE commands, generated by the best performing algorithms, which were rated as useful and novel, as reported by Murphy-Hill *et al.* [9]. It is also considerably less than in the study reported by Li *et al.* [19], with AutoCAD recommendations.[10]

Despite a relatively low recommendation acceptance rate, we observed a considerable increase in the number of newly executed commands, which followed the introduction of the ICRS GUI. Before our system started recommending commands, an average student from the experimental group (see [16]) executed around 56 distinct commands. By the end of the study, this number increased to 80. Conversely, the number of executed distinct commands by the students who were using only data collection tools, and did not receive recommendations, i.e., control group, remained almost the same in the second half of the study, which means that they basically stopped executing new commands, after less than 20 weeks. We can conclude that by providing recommendations, even if the acceptance rate may be low, the discovery rates increase considerably, when our ICRS is introduced.

The answers to the questions at the beginning of the third questionnaire show that our study participants had a positive opinion of the CoRe ICRS. As can be seen in Fig. 4, all participants liked the commands recommended by the system and 54% consider them relevant for their tasks, while others were undecided. Furthermore, as shown in Fig. 5, only 8% of the participants think that the system recommended too many bad commands, 15% are undecided, 69% disagree with the statement, and 8% strongly disagree.

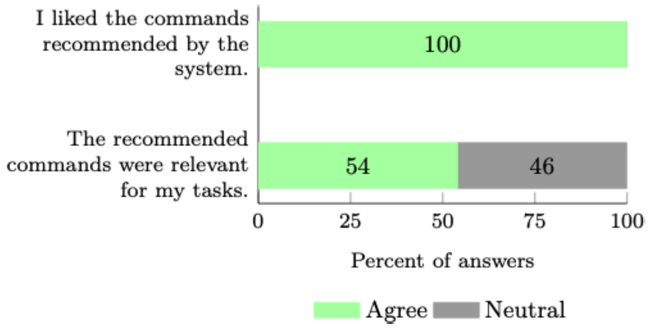


FIGURE 4. Agreement with the "I liked the commands recommended by the system" and "The recommended commands were relevant for my tasks" statements.

2) RQ2: HOW USERS LEARN NEW COMMANDS AND HOW MUCH THEY ARE AWARE OF HOW THEY LEARN THEM?

As already mentioned, we collected 53 answers to the first question in the second questionnaire, which is about the learning source for the commands that have been newly discovered and regularly used. However, five replies were: "Actually, I am not using this command". We excluded these answers from the analysis, since we only want to consider commands that the users did use, and they were also aware of the fact.

Amongst the remaining 48 replies, in 23 of them, which is 48%, the users state that the ICRS recommended the command. However, after reviewing automatically collected data about IDE-interaction, we could see that only 12 commands of those were actually recommended. The other 11 replies, which are wrong, were provided by 5 different participants. Consequently, we can conclude that at least some study participants are either unaware or oblivious to the source from which they learned new commands. The users seem to overestimate the influence of the ICRS and assume that they learn from ICRS even more than they actually do.

When it comes to other learning sources, 29% of the commands were recommended to the ICRS users by another person; 13% of the commands were discovered autonomously, when the users explored the Eclipse interface; 6% were discovered when a user noticed that someone else is using it; and 4% were discovered in a way that was not listed amongst possible answers. These proportions are visualised in Fig. 6.

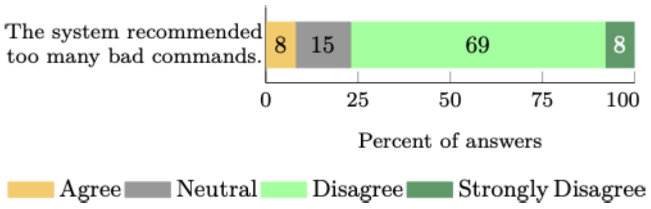


FIGURE 5. Agreement with the "The system recommended too many bad commands" statement.

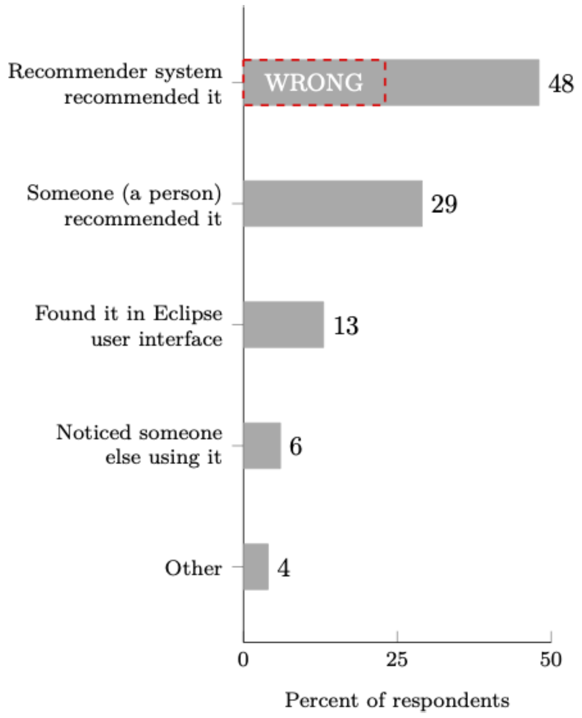


FIGURE 6. Learning sources for newly discovered and regularly used commands.

[10]To allow the comparison of the results obtained by Murphy-Hill et al. and Li et al., we had to calculate how many of all recommended commands were identified as useful and novel by recommendation recipients. In [9], the best performing algorithms were Most Popular and User-based Collaborative Filtering, which recommended 19.2% of such commands. We note that those recommendations were not delivered in a real-life setting, instead, they were presented orally by the researchers. In [19], Item-based Collaborative Filtering recommended 30.9% of commands rated as useful and novel.

We would like to add that during the study we detected 103 newly discovered and regularly used commands, amongst which 59 were not recommended, 35 were recommended by Most Popular, 8 by CoDis, and 1 by CNTX. This clearly shows that the commands that are used most frequently, after they are discovered, are either recommended by the Most Popular algorithm or they are discovered without the ICRS.

We assume that this is due to the fact that in IDEs there are commands that are often executed several times in a very short time period, in order to achieve the desired result, such as, Resume, Toggle Breakpoint, Step Into, Step Over, Select Next Word, Select Previous Word, Refresh, etc.; and there are others, which are usually executed only once, such as, Generate Getters and Setters, Build All, Toggle Breadcrumb, and Add Javadoc Comment. Consequently, even though all the listed commands were discovered during our study, it is clear that our requirement somewhat biased the data collection in a way that favoured the commands recommended by the algorithm that is based on the execution frequency.

3) RQ3: WHAT ARE THE REASONS FOR REJECTIONS OF RECOMMENDATIONS?

In the third questionnaire, we asked the participants why they did not use certain recommended commands. We received answers for 31 recommendations, but we excluded 6 answers, because the participants said that they actually think that they are using this command.

As can be seen in Fig. 7, most of the recommended commands have not been used because the recommendation recipient forgot about the command. This was the answer for 40% of the questions. In 28% of the cases, the recipients think that the command is not useful for their work. In 16% of the cases, the recipients do not know exact reasons for not using the command. In 12% of the cases, the recipients are already using other commands to do the same thing as the recommended command would do. And only in 4% of the cases, which represent one relevant answer in this survey, the recipient thinks that the recommended command is too hard to use. This particular command is Force Return, which is considered quite complex also by us.

It seems that the number of rejected recommendations could be decreased with some improvement of the GUI and the recommendation algorithm. As already mentioned, almost 82% of the recommended commands were never executed and we observe that each component is related to around 40% of these implicit rejections. Our hypothesis is that the acceptance rate could be enhanced considerably, by improving the quality of ICRSs, however, the theoretical limit is likely below 100%, since a notable part of the rejections remains unexplained.

4) RQ4: WHICH GUI PARTS ARE IMPORTANT AND USEFUL?

Fig. 8 shows which parts of the GUI are considered necessary and which useful for the evaluation of the quality of the command recommendation, by 13 participants who answered the third questionnaire. The participants selected the usage example as the most useful GUI part. The part that contains command name and description is considered necessary by 69% of the participants and useful by 77% of the participants. Finally, the explanation of the recommendation is considered the least important GUI part: only 8% consider it necessary and 31% consider it useful.

Additionally, when we look at the answers related to the very last question, which is about the missing information in the GUI, we can see that 12 out of 13 respondents to

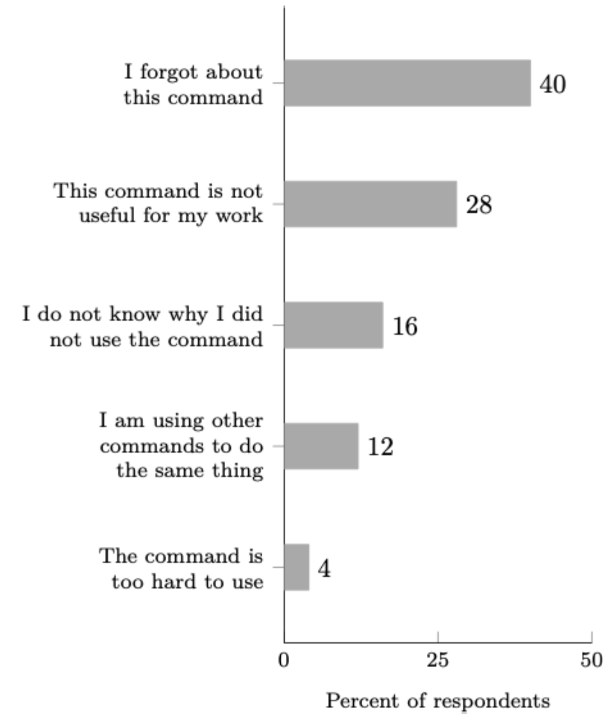


FIGURE 7. Reasons for recommendation rejection.

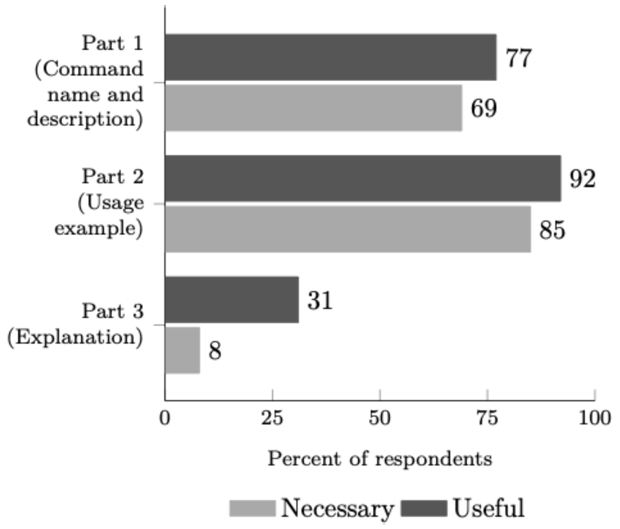


FIGURE 8. Results of the questions about the usefulness of the GUI parts.

the questionnaire think that all the necessary information is already provided. And one participant who thinks that some information is missing did not tell us which information that is. Since the results are in agreement with the findings of Gasparic *et al.* [13], it is safe to assume that providing command name, description, and usage example should suffice.

Furthermore, we studied the interaction with the GUI, as it was detected by our monitoring tools. In the last 14 weeks of the study, we detected 355 events that denote the opening of the recommendation presentation and 107 opening events of the recommendation explanation. Step-by-step video guides remained almost completely unused. We detected only 15 clicks on the link, by 6 users. If this number was higher, we could question the quality of the videos, however, based on the information we have, we can only conclude that after reading the command presentation the users were not willing to spend more time interacting with the system, apropos of the specific command. This is also confirmed by the observation that most of the videos were longer than a minute, but the median time spent for reading the content of one command recommendation presentation was less than 11 seconds.

107 recommendation explanation opening events refer to 87 distinct commands, which means that the users rarely look at the same recommendation explanation twice. This is yet another indicator that the attention span available to the ICRSs is very short.

Interestingly, the users who often open the explanation of the recommendation rationale consider it unnecessary and not useful for evaluating the recommendation quality. Conversely, the users who consider it useful are reading it very rarely: amongst 4 participants (i.e., 31%) who marked the explanation as useful, one opened 3 explanations, one opened 1 explanation, and two did not open any explanation. We can see from the logs of IDE-interaction events also that when the recommendation recipient opens the presentation, there is a higher chance that she will execute the command in the future than if the presentation was not opened. Moreover, if the recommendation explanation was opened, this chance is even higher. More precisely, if the recommendation presentation was opened, the chance of acceptance is 34% and only 10% otherwise. If the explanation was opened, the chance of acceptance is 42% and 19% otherwise. All that is in the agreement with one of the previous observations that the users are not the most reliable source of information. Still, we can conclude that providing recommendation explanation should not be considered as mandatory.

5) RQ5: WHEN ARE USERS INTERACTING WITH THE ICRS?

When we studied ICRS-interaction history logs, we noticed that the users tend to check several recommendations in a short time period. We treated the sequences of recommendation openings where two consecutive events were less than three minutes apart as one session. And we observed that 355 recommendation presentation openings occurred in only 83 such sessions. Hence, on average, more than 4 recommendations are opened in one ICRS-interaction session. There are 20 session when only one recommendation was opened and there are 16 sessions when more than 5 recommendations were opened.

We also analysed whether the users interact with the ICRS before, during, or after the work. We combined the IDE-interaction logs with the ICRS-interaction logs. We considered ICRS-interaction sessions that happened at least three minutes after the previous IDE-interaction event as sessions that are detached from the previous work, i.e., they could only occur *before* the student started working. ICRS-interaction sessions that happened at least three minutes before the following IDE-interaction event were considered as sessions that are detached from the following work, i.e., they could only occur *after* the student started working. Sessions for which both conditions were satisfied were considered as sessions that occurred *in isolation*. And the sessions for which none of the conditions was satisfied were considered as sessions that occurred *during* the work. Our results show that 37% of the sessions occur *before* the work, 17% occur *during*, and 8% *after*. 37% of the sessions occur *in isolation*.

These results indicate that only a small proportion of command recommendations had a potential to be executed soon after they were recommended. In our opinion, this means that even in a case of well-tailored *opportunistic* suggestions, the effect of the contextual relevance of the recommended commands on the recommendation acceptance rate would be small. Instead of focusing on the timing of actual recommendations from the perspective of their applicability in the given context, it would be worth considering improving their memorability.

Furthermore, as our results show, out of 1025 commands used by 19 volunteers for the first time after the introduction of the GUI, i.e., after April 13th 2017, 230 were executed within the first hour since the notification of newly available recommendations.[11] This is more than 22% of all new executions. On the other hand, only 35 of those commands were executed in the first three minutes. Because only one user received more than 6 such notifications,[12] in the period of 14 weeks, it is safe to assume that a higher frequency of notifications could lead to additional exploration of the IDE and even more executions of new (recommended) commands.

V. DISCUSSION AND FUTURE WORK

Knijnenburg and Willemsen [35] define user studies as small, observational, and used to iteratively improve the usability of RSs. In general, such studies offer a possibility to gather important evidence about a certain real-life phenomenon, which can be studied in-depth, within a setting that does not separate the phenomenon from its natural context,

[11] We excluded from the analysis 95 commands that were executed before the first notification was shown.

[12] The notifications were supposed to be visualised altogether six times, on the first IDE launch after a new set of recommendations was available. However, apparently, one user did not always close the notification dialogue, so it reappeared at the next IDE launch. We detected 10 dialogue openings for this particular user.

as laboratory experiments do [36]. Consequently, user studies, such as the one reported in this paper, provide valuable insights that can complement more quantitative empirical evidence originating from other types of research.

As our results show, the perceived recommendation quality of commands suggested by three different algorithms is high, despite the fact that the actual acceptance rates are relatively low. Students who installed our CoRe ICRS explored the IDE functionality autonomously and their command discovery rates increased considerably compared to the students who did not receive recommendations. ICRS users assigned credit to the system for learning more commands from it than they actually did, which also indicates that the users are not well aware of the learning sources for new IDE commands.

It seems that the acceptance rates could be considerably improved by better algorithms and also by a different GUI. As we found out, more than 80% of the recommended commands were never executed because the users forgot about them, the commands were not useful for the users' work, the users were using other commands to achieve the same result, or the commands were too hard to use.

Furthermore, our results show that opening of command recommendation presentation significantly increases the chance of recommendation acceptance and that more than 90% of the users think that it contains all the required information. It is also obvious that the users rarely test recommendations by executing commands. If that was not the case, the recommendation acceptance rate would be higher and the number of interactions with ICRS that happened in isolation or after the work would be lower.

We had a chance to analyse which parts of the recommendation presentation are to be included in the future ICRSs and we could compare the results of an online evaluation with the mock-up based prototype evaluation reported by Gasparic *et al.* [13]. Usage examples are considered the most important. They were marked as useful and as necessary by even a larger proportion of the participants than in the previous study. Since step-by-step video guides were viewed very rarely, it is safe to assume that only a picture of usage example is necessary to be included in the presentation.

Command name and description of its function are also important, but less than what we would expect, according to the previous research results. On the other hand, our results suggest that the explanation of the recommendation can be omitted. While it is anecdotal knowledge that recommendation explanations are important in RSs, in our live user study we found that it is not required; a similar conclusion applies for video guides. The fact that ICRSs do not need to be transparent, opens the door to a whole set of machine learning models and algorithms that perform well in other domains, but depend on a large number of parameters, which makes it hard to explain why certain commands are recommended. Examples would include deep-learning algorithms, which are very popular nowadays [37].

Furthermore, our analysis of the observed interactions with the GUI confirmed that the users prefer to assess recommendations when they are not working. Moreover, they generally open more than one command recommendation presentation in a single session. Hence, it seems that *global* recommendations are more suitable for software developers than *opportunistic* ones, even when the recommendations are context-aware (as it was in our experiment). It also shows that frequently forced interruptions may not fit ICRSs. However, when we asked our students why they did not execute certain recommended commands, the most common answer was that they forgot about them, which indicates that there is room for developing new methods for improving the user's awareness of available recommendations. Maybe, it is worth recommending the same command more than once, if the system is confident that this is a useful recommendation. Hence, digital nudges for encouraging developer actions may offer a viable solution [38].

To summarise our findings, researchers and IDE vendors should create command recommendation presentations that include the description of the command and a good usage example, while they focus on new algorithms and tailoring of command recommendation timing to further explore the domain of ICRSs. We are confident that these two aspects are strongly related to contextual information, which may play even a more crucial role in the future.

Context is any information that can be used to characterise the situation of an entity that is considered relevant to the interaction between a user and an application [39]. We distinguish between two types of context, namely, the context of the past IDE command executions and the context of the software developers in which they work and execute (recommended) commands. The first type is the most natural contextual information in ICRSs. It is the history of executed commands, often including the information about the timing[13] of these executions. This information is required in order to build the catalogue and avoid recommendations of known commands. Despite the simplicity of such data, it is powerful enough in certain situations. For example, in our study, the algorithm that only counts command executions and recommends the most popular unknown commands achieves much higher acceptance rate than the more sophisticated solutions: CoDis and CNTX [16]. If one wants to quickly deploy a system that generates recommendations, while collecting additional contextual data, or one is targeting unskilled IDE users, starting with a history of command executions is worthwhile. Additionally, there are publicly available data sets of command executions that can bootstrap such a process.

When it comes to additional contextual data, their accumulation is rather expensive and developers may be reluctant to share it [13]. Still, we believe that there is a lot of potential in advanced context modelling and in recommendation algorithms that can handle such information better than simple collaborative filtering, which generally suffices

[13]The timing of command executions can either refer to a raw timestamp, command's position in a sequence of executions, session-related association, or time of the day.

in other domains. In ICRSs, it seems that developers' activities represent the most meaningful information. In fact, a developer would use different commands when editing or debugging, for example. Damevski *et al.* [7] and our research group [8] dug into this problem and some promising results were obtained: in offline settings, new context-based algorithms performed better than existing algorithms, according to the selected metrics. Conversely, the usage of other IDE usage data and development contexts, such as, developers' knowledge, information about opened projects and screen content, or reports on unsuccessful executions, is still largely unexplored.

We also think that context would be highly useful for correctly timing when the recommendations should be offered to the user. Damevski *et al.* [7] already showed that it is possible to predict future behaviour, using context, hence, we see a great potential in this relatively unexplored domain. In fact, by tailoring the list of recommendations to activities that are predicted to happen in the near future, one can gain some benefits of the *opportunistic* recommendation strategy, in particular, a higher relevance of recommendations in the given situation, which may lead to better memorisation of recommendations, even when providing *global* suggestions.

VI. CONCLUSION

Software developers learn about available tools in different ways. For instance, during interactions with peers, which is an effective, but infrequently used technique [31]; or from paper newsletters posted in Google restrooms, where the effectiveness of learning depends on different factors, such as tool applicability and name memorability [40]. They can also learn new tools from software applications, in particular, IDE command recommender systems [16].

In this paper, we have introduced and discussed the concept of IDE command recommendation and the results of a long-term study where our recommender was used by a group of students. We believe that the presented analysis can help to better tackle this problem in the future.

To the best of our knowledge, this is the largest deployment of an ICRS that is not integrated in IDEs available on the market. We studied the interactions with the IDE and ICRS GUI, using automatically collected data, and we used questionnaires to obtain additional information about ICRS users' opinions and reasoning. We answered five research questions related to the IDE command recommendation delivery, an aspect that is often neglected in this domain.

We plan to further explore the timing of command recommendations and we hope that the research community will continue with ICRS research as well. We believe that this interesting functionality will someday be widely accepted and put into practice.

REFERENCES

[1] P. Bourque, and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge*. Washington, DC, USA: IEEE Computer Society Press, 2014.
[2] A. Ahmed, *Software Project Management: A Process-Driven Approach*. Bengaluru, India: Taylor & Francis, 2011.
[3] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Softw.*, vol. 27, no. 4, pp. 80–86, Jul./Aug. 2010.
[4] T. Grossman, G. Fitzmaurice, and R. Attar, "A survey of software learnability: Metrics, methodologies and guidelines," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 2009, pp. 649–658.
[5] E. Murphy-Hill, "Continuous social screencasting to facilitate software tool discovery," in *Proc. 34th Int. Conf. Softw. Eng.*, 2012, pp. 1317–1320.
[6] G. Fischer, "User modeling in human–computer interaction," *User Model. User-Adapted Interact.*, vol. 11, pp. 65–86, Mar. 2001.
[7] K. Damevski, H. Chen, D. C. Shepherd, N. A. Kraft, and L. Pollock, "Predicting future developer behavior in the IDE using topic models," *IEEE Trans. Softw. Eng.*, vol. 44, no. 11, pp. 1100–1111, Nov. 2018.
[8] M. Gasparic, T. Gurbanov, and F. Ricci, "Context-aware integrated development environment command recommender systems," in *Proc. 32nd IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Oct. 2017, pp. 688–693.
[9] E. Murphy-Hill, R. Jiresal, and G. C. Murphy, "Improving software developers' fluency by recommending development environment commands," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng.*, 2012, pp. 1–11.
[10] M. Schmidmaier, Z. Han, T. Weber, Y. Liu, and H. Hußmann, "Real-time personalization in adaptive IDEs," in *Proc. Adjunct Publication 27th Conf. User Modeling, Adaptation Personalization*, 2019, pp. 1–6.
[11] D. Silva, R. Terra, and M. T. Valente, "Recommending automated extract method refactorings," in *Proc. 22nd Int. Conf. Program Comprehension*, 2014, pp. 146–156.
[12] S. Zolaktaf and G. C. Murphy, "What to learn next: Recommending commands in a feature-rich environment," in *Proc. IEEE 14th Int. Conf. Mach. Learn. Appl.*, Dec. 2015, pp. 1038–1044.
[13] M. Gasparic, A. Janes, F. Ricci, G. C. Murphy, and T. Gurbanov, "A graphical user interface for presenting integrated development environment command recommendations: Design, evaluation, and implementation," *Inf. Softw. Technol.*, vol. 92, pp. 236–255, Dec. 2017.
[14] S. Stuckemann, "Vignelli-automated design guidance for developers," Dept. Comput., Imperial College London, London, U.K., Tech. Rep., 2015.
[15] P. Viriyakattiyaporn and G. C. Murphy, "Improving program navigation with an active help system," in *Proc. Conf. Center Adv. Stud. Collaborative Res.*, 2010, pp. 27–41.
[16] M. Gasparic, T. Gurbanov, and F. Ricci, "Improving integrated development environment commands knowledge with recommender systems," in *Proc. 40th Int. Conf. Softw. Eng., Softw. Eng. Educ. Training*, 2018, pp. 88–97.
[17] E. Murphy-Hill and G. C. Murphy, "Recommendation delivery," in *Recommendation Systems in Software Engineering*. Berlin, Germany: Springer-Verlag, 2014, pp. 223–242.
[18] T. Bulmer, L. Montgomery, and D. Damian, "Predicting developers' IDE commands with machine learning," in *Proc. IEEE/ACM 15th Int. Conf. Mining Softw. Repositories*, May/Jun. 2018, pp. 82–85.
[19] W. Li, J. Matejka, T. Grossman, J. A. Konstan, and G. Fitzmaurice, "Design and evaluation of a command recommendation system for software applications," *ACM Trans. Comput.-Hum. Interact.*, vol. 18, no. 2, pp. 1–35, Jun. 2011.
[20] N. Tintarev and J. Masthoff, "Explaining recommendations: Design and evaluation," in *Recommender Systems Handbook*. New York, NY, USA: Springer, 2015, pp. 353–382.
[21] D. C. Mcfarlane and K. A. Latorella, "The scope and importance of human interruption in human-computer interaction design," *Hum.-Comput. Interact.*, vol. 17, no. 1, pp. 1–61, Mar. 2002.
[22] M. Gasparic, G. C. Murphy, and F. Ricci, "A context model for IDE-based recommendation systems," *J. Syst. Softw.*, vol. 128, pp. 200–219, Jun. 2017.
[23] M. Gasparic and F. Ricci, "Should context-aware IDE command recommendations always be presented in-context or not?" in *Proc. AWARE Workshop*, 2017.
[24] F. Linton and H.-P. Schaefer, "Recommender systems for learning: Building user and expert models through long-term observation of application use," *User Model. User-Adapted Interact.*, vol. 10, pp. 181–208, Jun. 2000.
[25] J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice, "CommunityCommands: Command recommendations for software applications," in *Proc. 22nd Annu. ACM Symp. User Interface Softw. Technol.*, 2009, pp. 193–202.

[26] M. A. A. Khan, V. Dziubak, and A. Bunt, "Exploring personalized command recommendations based on information found in Web documentation," in *Proc. Int. Conf. Intell. User Interfaces*, 2015, pp. 225–235.

[27] M. Wiebe, D. Y. Geiskkovitch, and A. Bunt, "Exploring user attitudes towards different approaches to command recommendation in feature-rich software," in *Proc. Int. Conf. Intell. User Interfaces*, 2016, pp. 43–47.

[28] F. Ricci, "Recommender systems: Models and techniques," in *Encyclopedia of Social Network Analysis and Mining*. New York, NY, USA: Springer-Verlag, 2014, pp. 1511–1522.

[29] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proc. ACM/SIGSOFT Int. Symp. Found. Softw. Eng.*, 2006, pp. 1–11.

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

[31] E. Murphy-Hill, D. Y. Lee, G. C. Murphy, and J. Mcgrenere, "How do users discover new tools in software development and beyond?" *Comput. Supported Cooperat. Work*, vol. 24, no. 5, pp. 389–422, Oct. 2015.

[32] V. Venkatesh, M. G. Morris, G. B. Davis, and F. D. Davis, "User acceptance of information technology: Toward a unified view," *MIS Quart.*, vol. 27, no. 3, pp. 425–478, 2003.

[33] B. P. Knijnenburg, M. C. Willemsen, Z. Gantner, H. Soncu, and C. Newell, "Explaining the user experience of recommender systems," *User Model. User-Adapted Interact.*, vol. 22, nos. 4–5, pp. 441–504, Oct. 2012.

[34] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell, *Experimentation in Software Engineering*. Berlin, Germany: Springer-Verlag, 2012.

[35] B. P. Knijnenburg and M. C. Willemsen, "Evaluating recommender systems with user experiments," in *Recommender Systems Handbook*. New York, NY, USA: Springer, 2015, pp. 309–352.

[36] R. K. Yin, *Case Study Research: Design and Methods*. Thousand Oaks, CA, USA: SAGE Publications, 2009.

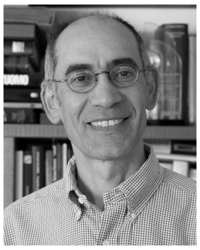
[37] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53040–53065, 2019.

[38] C. Brown, "Digital nudges for encouraging developer actions," in *Proc. Int. Conf. Softw. Eng., Companion*, 2019, pp. 202–205.

[39] A. K. Dey, "Understanding and using context," *Pers. Ubiquitous Comput.*, vol. 5, no. 1, pp. 4–7, 2001.

[40] E. Murphy-Hill, E. K. Smith, C. Sadowski, C. Jaspan, C. Winter, M. Jorde, A. Knight, A. Trenk, and S. Gross, "Do developers discover new tools on the toilet?" in *Proc. Int. Conf. Softw. Eng.*, 2019, pp. 465–475.

MARKO GASPARIC received the Ph.D. degree in computer science from the Free University of Bozen–Bolzano. He was the product manager and the development team leader of applications used by more than million paying customers. He is currently a Researcher with the Faculty of Electrical Engineering and Computer Science, University of Maribor. He has published in premier software engineering conferences and journals. His professional interests include software development, programming, agile techniques, artificial intelligence, and blockchain.



FRANCESCO RICCI has established a reference point for the research on recommender systems in Bolzano. He has been active with this community as the President of the Steering Committee of the ACM Conference on Recommender Systems, from 2007 to 2010. He was previously a Senior Researcher and the Technical Director of the E-commerce and Tourism Research Lab (eCTRL), ITC-IRST, Trento, Italy, from 2000 to 2006. From 1998 to 2000, he was a System Architect with the Research and Technology Department (process and reuse technologies), Sodalia S.p.A. He is a Full Professor and the Dean of the Faculty of Computer Science, Free University of Bozen–Bolzano. He has participated in several international research projects such as RECOM (funded by Deutsche Telekom), etPackaging (funded by ECCA), European Tourist Destination Portal (funded by European Travel Commission), Harmoten (funded by IST), and DieToRecs (intelligent recommendation for tourist destination decision making, funded by IST). He is the author of more than one-hundred and fifty refereed publications. According to Google Scholar, he has an H-index of 52 and around 17 000 citations. He was the Co-Editor of the *Recommender Systems Handbook* (Springer 2011 and 2015).