Improving Integrated Development Environment Commands Knowledge With Recommender Systems

Marko Gasparic, Tural Gurbanov, Francesco Ricci
Free University of Bozen-Bolzano
Dominikanerplatz 3, 39100 Bolzano, Italy
m.gasparic@gmail.com, tural.gurbanov@inf.unibz.it, francesco.ricci@unibz.it

ABSTRACT

Development tools have an impact on software engineers' productivity and quality of software construction. We believe that it is crucial to teach future software engineers how to exploit integrated development environment functionality, if we want to encourage the effective application of software development principles and practices. Our research shows that recommender systems can be deployed to improve integrated development environment knowledge of computer science students by automatically suggesting new and useful commands, such as buttons and shortcuts that execute different functions. While previous work focused on optimizing the algorithmic predictive capability of a recommender to identify the commands that the users will eventually use, we have addressed a set of research questions related to the overall acceptance of a complete recommender system in a real-life setting. The evaluation results show that a command recommender system can be well accepted by computer science students. In particular, when students are supported by such a system, they use a considerably larger set of commands available in their development environment. Moreover, the results show that the highest acceptance rate and the usefulness score were achieved by a non-personalized, popularity-based algorithm, while the most novel commands were suggested by a context-aware algorithm.

ICSE-SEET'18, May 27-June 3 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5660-2/18/05…$15.00
https://doi.org/10.1145/3183377.3183386

1 INTRODUCTION

The programmers' knowledge about the capability of the used development tools impacts their productivity and the quality of software construction [12, 25, 27, 30]. One reason for this is that tools make software development more systematic and they also "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" [1, pp. 368]. Popular applications that serve the needs of software developers, by bringing together multiple tools to create and manipulate software project artifacts, are integrated development environments (IDEs).

Previous research indicates that a lack of knowledge is a common reason for potentially useful functionality not being used [3, 11]. Furthermore, Murphy-Hill [18] discovered that Eclipse[1] users, on average, execute less than 50 different commands out of 1100; where a command corresponds to a menu button or a shortcut that executes a function, such as Undo or Organize Imports, amongst many others.

According to Murphy-Hill et al. [20], students and professional software developers most often learn about new tools by coincidentally encountering them, during the exploration of their IDE. However, while this is the most common way to discover new tools, it is also ineffective, because this somewhat random process does not guarantee that the users learn what they need to know for their work; and, as Murphy-Hill et al. noticed, it is not likely that a tool discovered in this way will be used again. On the other hand, the most effective way to discover new tools is to learn about them from peers, which, however, occurs very infrequently [20].

We conjecture that improving the command knowledge in high-functionality applications, such as IDE, is a suitable task for a recommender system (RS). RSs are personalised information search and filtering tools that suggest useful items [22]. They are conceived to support individuals who lack sufficient time or knowledge to evaluate a large number of alternative options that may be available [21, 23].

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

According to Fischer [4], unless some active mechanism which suggests specific unknown functionality to the users is introduced, it is actually very hard for them to learn new relevant functionality that they are not at least vaguely familiar with. Furthermore, we stress that it is not expected that generic users would access the full set of the provided functionality because most of them are not application experts and they often only want to efficiently accomplish the task at hand. Hence, the recommendations have to be tailored, at least to some extent. Additionally, users should not be annoyed with functionality recommendations which are already known, thus, non-personalised mechanisms, such as "Did You Know" and "Tip of the Day", do not suffice [4].

In short, an IDE command RS should aim at recommending useful commands that would be difficult to discover without the help of the RS. As stated by Matejka et al. [16]: commands that are not useful for users can be considered as poor recommendations and recommendations of commands that users already know are unnecessary.

We believe that it is crucial to teach future software engineers how to exploit integrated development environment functionality, if we want to encourage effective application of software development principles and practices. Moreover, we show in this paper that the breadth of the IDE functionality used by first year bachelor students increases considerably if they use an IDE command RS.

The idea to apply RSs to suggest IDE commands is not novel [5, 7, 19, 29, 32], as we discuss in Section 2. Nevertheless, we still do not know what is the actual effect of such a system in a real-life setting: previous research findings based on the studies performed in offline settings are inconclusive, with respect to the actual recommendations' utility, and only one very limited study was conducted online. We addressed these gaps with an observational study, experiments in a real-life setting, and an offline study based on expert evaluation, which answered the following research questions:

RQ1 What is the effect of an IDE command RS on the command discovery rates of students?

RQ2 How well accepted, novel, and useful are the recommendations of different algorithms?

RQ3 What is students' opinion about the recommendations displayed by the RS?

We describe our implementation of an IDE command RS in Section 3. We present the adopted research method in Section 4. We list relevant validity threats, present the results, and discuss those results in Section 5, 6, and 7, respectively. And we conclude the paper in Section 8.

2 RELATED WORK

There are two main strategies to recommend commands in high-functionality applications [16]. In the first, which is called "*global* suggestions", recommendations are based on the full user's command usage history and are useful in a long-term perspective. Hence, with these recommendations, it is possible to maximize the overall utility of the RS. In the second strategy, which is called "*opportunistic* suggestions", recommendations are particularly relevant in the specific situation in which the developer is at the moment of the recommendation. The main advantage of *opportunistic* suggestions is that they usually allow instant execution of the (accurately) recommended commands; hence, the user may immediately recognize the benefit of the suggestions. However, caring only about the current user's context may end up with myopic suggestions [10]. For this reason, in our work, we are focusing on *global* suggestions.

A set of alternative algorithms for generating *global* IDE command recommendations has been proposed, and to some extent evaluated, in [5], [19], and [32]. The algorithms that are particularly relevant for our study are:

- Most Popular [19], which is a baseline algorithm that recommends the most frequently executed commands that are not already used by the recipient;[2]
- CoDis [32], which combines the information about the co-occurrences of commands in the same session with command discovery patterns;[3] and
- CNTX [5], which is the most recent algorithm, and it is based on the analysis of similarities of contexts in which recommendation recipients work and in which commands are executed.[4]

Other algorithms for *global* IDE command recommendations are based on command popularity, collaborative filtering, or command discovery patterns. In our study, we considered the

[2]Most Popular is a non-personalized algorithm. It first sorts all commands according to the overall execution frequency, within the entire population of users, then it removes those that a recommendation recipient already knows, and finally it selects recommendations with the highest scores, from the remaining list of commands.

[3]CoDis analyzes which commands are often executed in the same work session and performs a sequential pattern mining that models the process of command discovery. Extracted co-occurring commands and discovery patterns are used to identify which commands are used with or discovered after the commands that the recipient is using. CoDis uses weighting scheme that combines the two scores, based on the elapsed time between recommendation generation and last activity.

[4]CNTX observes contexts in which commands are usually executed and contexts in which recommendation recipient often works. It recommends the commands with the highest context similarity. The context representation is based on the model presented in [9].

above mentioned algorithms because Most Popular is often used for benchmarking RSs, also in other domains, and it performed well in the online study reported by Murphy-Hill et al. [19], even though it is a very basic algorithm; CoDis and CNTX performed best in the offline evaluations conducted by Zolaktaf and Murphy [32] and Gasparic et al. [5].

We observe that it is hard to draw reliable conclusions about the effective quality of the considered RS algorithms from the previous studies, because the only online evaluation of *global* IDE command RS algorithms, which is reported in [19], included a small set of participants.[5] Additionally, the recommendations were not delivered by a system, but verbally by the researchers, which likely induced biases related to "hypothesis guessing" and "experimenter expectancies" [31]. Moreover, we still do not know how software developers react to command recommendations delivered in a real-life setting, by a complete RS. For example, we expect a different behavior when the users are interacting with a system for several weeks, compared to a case when they are interacting for the first time with a researcher. Moreover, the quality of the graphical user interface (GUI) can have a big impact on recommendation acceptance and the working context can influence perception of recommendations as well.

On the other hand, the offline evaluation method suggested by Li et al. [15],[6] which is typically used to evaluate command recommendation algorithms in high-functionality applications—including IDEs—only measures whether the algorithms predict correctly which command is discovered next by the user. However, we believe that optimizing the algorithms for making good *predictions*, instead of making good *recommendations*, leads to redundant, not novel, and not very useful command recommendations. For example, recommending Save command after editing, Paste command after Copy, or simply always recommending Undo command, could eventually lead to highly accurate *predictions*, but it would not increase developers' knowledge about the available IDE functionality. In fact, as the studies reported in this paper and by Murphy-Hill et al. [19] show, the best performing algorithms according to the offline evaluations are not always recommending the most useful commands.

It seems that the research on command recommendations is mimicking the research on traditional RSs from a decade ago, when an extensive focus on recommendations that were (highly) accurate according to standard offline metrics led to recommendations that were not very useful in practice [17]. We conjecture that measuring the predictive power of an algorithm is not the right approach. Instead, a command RS should aim at recommending useful commands that would be difficult to discover without the system support.

Furthermore, when it comes to deploying a complete IDE command RS, it is also necessary to design an effective GUI. The only existing GUI for *global* IDE command recommendations is presented in [7] and [8]. This GUI is independent of the underlying algorithm, it was built according to a set of guidelines identified in the literature, and it was shown to be well accepted by the target population. Hence, it is suitable for the purposes of our study. However, we note that the qualities of this GUI were assessed with interviews and paper-based prototypes in an artificial setting.

Besides the presented research work related to *global* IDE command RSs, there exist also two systems that provide *opportunistic* suggestions, namely Spyglass [29], which recommends navigation commands when it detects suboptimal navigation over source code, and Vignelli [28], which detects design flaws in the code and recommends suitable refactoring commands. However, these RSs are only able to recommend a limited set of commands and their GUI design cannot be applied to all types of IDE commands [7]. Hence, we could not use them in our study.

3 IDE COMMAND RS IMPLEMENTATION

Our target IDE is Eclipse and the RS is primarily implemented in Java. To collect the data needed to generate recommendations, we use Eclipse UDC[7] and a special version of Mylyn [13].[8] When Eclipse is running, these two monitoring tools automatically collect the data in the background. When a command is executed, a file is modified, the focus changes, a new plugin is installed, etc., the log is stored in the local

[5] Murphy-Hill et al. [19] recruited four experts and nine novices, for whom they generated 8 recommendations with each of 8 evaluated algorithms. The authors do not report for how long they were collecting the data before they generated recommendations. The participants had to rate the novelty and the usefulness of the commands. The results show that the best performing algorithms were Most Popular and User-based Collaborative Filtering, which recommended 19.2% of commands that were rated as useful and novel.

[6] The method is called "k-tail evaluation" [15]. The interaction history logs are split in the set of command executions that occurred before the user started using the last k commands (i.e., training set) and in the set of commands that were executed after that moment (i.e., test set). Then, the algorithm score is calculated as $\frac{1}{|U|}\sum_{u \in U} hit_u$, where U is the set of all users u and hit_u is a binary variable equal to 1 when the set of recommended commands includes at least one k command and equal to 0 if the recommended commands include none of the k commands. Usually, the value of k is 1, which means that the algorithms are aiming to guess for each user the last discovered command included in the interaction history log.

[7] http://eclipse.org/epp/usagedata

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

workspace. In parallel, another Eclipse plugin is periodically uploading new logs to our database.[9]

Once the logs of the user's interaction with the IDE are moved from the clients to the database, the RS analyzes them, exports the data required for the recommendation generation, executes the RS algorithms, and stores the recommendations back into the database. To automate these tasks, we implemented platform independent Java applications. As already mentioned, the algorithms included in our RS are: Most Popular, CoDis, and CNTX.[10]

There is a two-way communication channel between the RS and the recommendation recipients: the RS informs the users when new recommendations are available and the users can visualize or hide a list of recommendations, open and close the description of a specific command, open a step-by-step video guide in a web browser, or open and close the explanation of the recommendation. Examples of the implemented GUI components are shown in Fig. 1.

When the recommendations are available, the GUI, which is implemented as an Eclipse plugin and is periodically checking for new recommendations, downloads them to the local workspace. In that way, the GUI can access recommendations promptly, also when there is no internet connection. When the recommendations are downloaded, the GUI shows a dialog in Eclipse (see Nr. 3, Fig. 1), which informs the recipient that new recommendations are available.

If the Recommendation List view (see Nr. 2, Fig. 1) is closed, the user can open it by selecting the "Show" button in the dialog—which notifies the presence of new recommendations—or by clicking on the RS icon in the toolbar (see Nr. 1, Fig. 1).[11] Next to the command, in the Recommendation List view, an icon also shows whether the recommendation has been read. When a user clicks on a recommendation in the list, the selected command is presented in the Recommendation view (see Nr. 4, Fig. 1), which is similar to the one suggested by Gasparic et al. [7, 8].

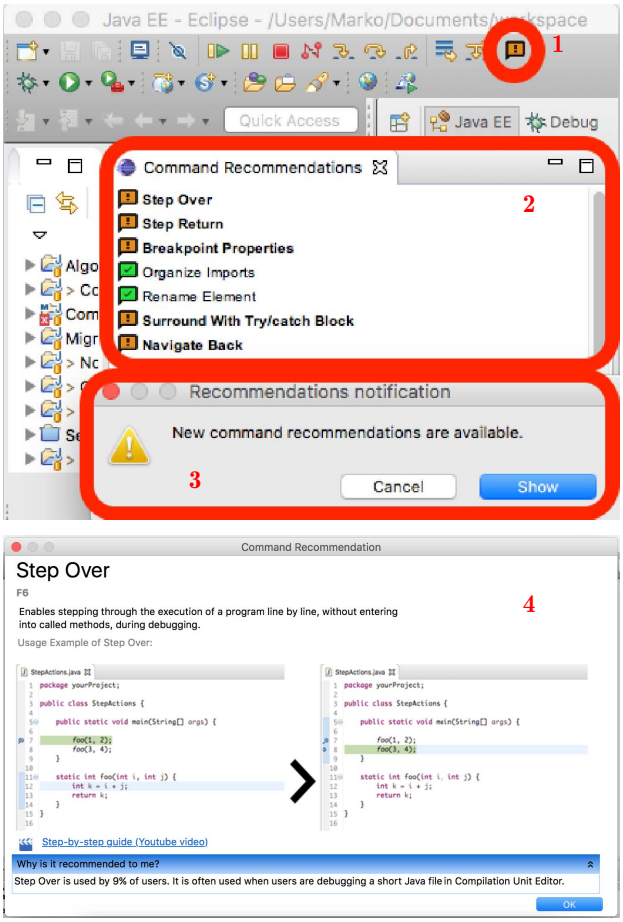

Figure 1: RS toolbar icon (1), Recommendation List view (2), Notification dialog (3), and Recommendation view (4).

We note that the user's interaction with the RS GUI is recorded and uploaded in a similar way as the interaction with the IDE. The full analysis of that interaction is out of the scope of this paper, but we plan to study it in the future.

All custom Eclipse plugins and collected data are available at: https://sites.google.com/site/mgasparic/experiment.

4 RESEARCH METHOD

To evaluate the quality of the complete RS and assess its effect on the behavior of the users, we invited first year computer science students at Free University of Bozen-Bolzano, to participate in our study. The majority of the study participants installed the monitoring tools in October 2016; others were invited to join the study at the beginning of March 2017. To participate in the study, the students had to install

[9] We note that the data used by the system to generate recommendations are collected for all the users who use an IDE with the monitoring tools installed, even if they do not receive recommendations. The data collection is anonymous and the event logs are associated with user identifiers, which are stored in the workspace of each user.

[10] For the Most Popular algorithm, we used the implementation of Murphy-Hill et al. [18], available at: https://github.com/ubc-cs-spl/command-recommender-generator. For CoDis, we used the implementation of Zolaktaf and Murphy [32], which we received directly from the authors. And for CNTX, we used the implementation of Gasparic et al. [5], available at: https://gitlab.inf.unibz.it/tural-gurbanov/ide_rs.

[11] The icon is a message cloud that can have three colors: red, when none of the new recommendations has been read or there are many unread recommendations; orange, when there are some unread recommendations; and green, when all recommendations have been read.

a custom version of Eclipse IDE, which already included the necessary monitoring tools. We started the data collection on October 12^{th} 2016 and stopped it on July 19^{th} 2017; hence, the entire data set used in the evaluation contains event logs collected during a 40 weeks period. On April 13^{th} 2017, we asked the students to install the GUI plugin for recommendations visualization. During the following 6 weeks, on every Thursday morning, each participant who installed the GUI plugin received top-5 recommendations.

The recommendations were based on the data collected between October 2016 and Friday evening before the Thursday on which recommendations were available. The delay of six days was necessary because the execution of the algorithms required some time and then we also had to manually prepare the descriptions, usage examples, and video tutorials for the commands that have not been recommended yet. We note that the manual generation of the content was particularly time-consuming at the beginning, however, because this task has to be performed only once for each distinct command, the workload decreased considerably over time.

During the entire study, we monitored the usage of the IDE commands and the interaction with the RS GUI. Additionally, to answer the research questions defined in Section 1, we used different supplementary scientific tools and methods; they are summarized in Tab. 1 and discussed in turn.

4.1 What is the effect of an IDE command RS on the command discovery rates of students?

To answer RQ1, we conducted an experiment, which can be classified as "one factor with two treatments", according to Wohlin et al. [31]. The factor in our experiment was the RS GUI and the two treatments were the installation or not installation of the GUI. We limited our analysis to the students who started interacting with the IDE on the first week of the study and were generating usage data also after April 13^{th} 2017, when the RS GUI was available for the installation. The application of the treatment was quasi-random, based on the voluntary selection of the group: students who installed the GUI received the first treatment and others received the second treatment, i.e., they served as a control group.

We adopted Pearson's χ^2 test to check whether the difference in the number of discovered commands between the two groups is statistically significant. The null hypothesis used in our test is: the number of discovered commands after April 13^{th} 2017 is independent of the installation of the RS GUI. The alternative hypothesis is: the number of discovered commands depends on the GUI installation.

Table 1: Research questions and corresponding methods.

Research question	Research tools and methods	Statistical tests
What is the effect of an IDE command RS on the command discovery rates of students?	Experiment ("one factor with two treatments")	Pearson χ^2
How well accepted, novel, and useful are the recommendations of different algorithms?	Experiment ("one factor with more than two treatments"), Questionnaire, Expert-based evaluation	Mann-Whitney, Pearson χ^2
What is students' opinion about the recommendations displayed by the RS?	Questionnaire	/

4.2 How well accepted, novel, and useful are the recommendations of different algorithms?

To answer RQ2, we studied the acceptance rate, novelty, and usefulness of the recommendations generated by different algorithms that appeared in the IDEs of the study participants.

First, we conducted an experiment, which can be classified as "one factor with more than two treatments" with "randomised complete block design", according to Wohlin et al. [31]. In such an experiment, each subject uses all treatments, which are assigned in a random order. The factor in our experiment was the algorithm used for generating command recommendations and the three different treatments were: Most Popular, CoDis, and CNTX. During the experiment, the participants were receiving recommendations from each algorithm for two consecutive weeks, without knowing which algorithm generated recommendations. The order of the algorithms was random, but we assured that there were only small differences between the numbers of participants who received recommendations in the alternative orders.

In our analysis, we considered a recommendation as accepted if the command was executed at least once by the end of the study and as rejected otherwise. This is a common approach also used in the previous research. We calculated overall and individual acceptance rates of commands that were visualized in the recommendation lists of students, i.e., at least the names of the recommended commands appeared in the IDEs of the recipients. We note that we excluded

the recommendations for which the first execution occurred before the visualization, because in this case the recipient certainly discovered the command without the help of our RS. To check whether the differences of the acceptance rates between the algorithms are statistically significant, we used Mann-Whitney's test. For each pair of algorithms, the null hypothesis is: the recommendation acceptance rate is independent of the algorithm. The alternative hypothesis is: the acceptance rate depends on the algorithm.

Furthermore, when the participants closed the Recommendation view related to a specific command (see Nr. 4, Fig. 1), we asked them: "Please tell us how familiar were you with '[Command name]' command, before it was recommended", and we provided a set of possible answers, which is based on the taxonomy proposed by Anderson-Meger [2]:

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

We used this question to measure the novelty of recommended commands. To check whether the differences between the algorithms are statistically significant, we used Pearson's χ^2 test. For each pair of algorithms, the null hypothesis is: the novelty of recommended commands is independent of the algorithm. The alternative hypothesis is: the novelty depends on the algorithm.

Finally, to assess whether the RS algorithms are recommending useful commands, we conducted an expert-based evaluation. We extracted the set of distinct commands that have been visualized in the IDEs of the experiment participants. Then, for each command, two authors of this paper expressed their agreement on a five-point Likert scale with the following statement: "First year computer science students should find the command useful in their work." Both experts evaluated the commands individually and without knowing which commands were recommended by which algorithm. We assigned a numeric value to each point on the Likert scale, by making "Strongly agree" equal to 2, "Agree" equal to 1, "Neutral" equal to 0, "Disagree" equal to -1, and "Strongly disagree" equal to -2. We used the average of the two judgments as the ground truth. To check whether the differences between the algorithms are statistically significant, we used Mann-Whitney's test. For each pair of algorithms, the null hypothesis is: the usefulness of recommended commands is independent of the algorithm. The alternative hypothesis is: the usefulness depends on the algorithm.

4.3 What is students' opinion about the recommendations displayed by the RS?

To answer RQ3, as suggested by Knijnenburg et al. [14], we asked the participants to mark on a five-point Likert scale how strongly they agree with the following three statements:

- "I liked the commands recommended by the system";
- "The recommended commands were relevant for my tasks"; and
- "The system recommended too many bad commands".

The participants were invited to answer this questionnaire one week after the last set of recommendations was available. We delayed the time when the questionnaire was made available to give the participants enough time to download and examine the final recommendations. Remember that the participants were not aware that we used different algorithms, hence, they were evaluating the system as a whole. Based on the answers to these three questions we could assess whether they generally liked recommendations or not.

5 VALIDITY THREATS

Conclusion validity is related to the correctness of the conclusions about the relations between the treatment and the outcomes of the experiment [31]. In our case, the main threat to the conclusion validity is a relatively small sample size: 28 participants were included in the experiment related to RQ1, 19 in the experiment related to RQ2, and 13 in the study related to RQ3. Thus, the ability to reveal correct patterns in the data is decreased. To mitigate this threat and to ensure that our data sets are sufficiently large for reliable conclusions related to the results of statistical tests, we conducted post-hoc power analyses with significance level 0.05. Still, the results of the RQ3 have to be taken with caution.

Internal validity is related to the correctness of the causal influences that affect the observed variables, without researcher's knowledge [31]. In our case, because the treatments were applied at different times and in a different sequence, it is possible that the history and the maturation of the subjects affected the experimental results; for example, user's trust in the system may have been negatively affected by initial bad recommendations. To mitigate this threat, we used a control group when evaluating the effect of the RS on the command discovery rate. Additionally, we decreased this threat by randomly selecting the sequence of the treatments for experiment participants. We assume that different effects would average out in the entire sample, but we were not able to control or measure the effects of history and maturation.

The second internal validity threat is related to the selection of the experiment 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 a better knowledge of IDE functionality than other students, who were also invited to participate, but refused the invitation.

Construct validity is related to the correctness of the results generalisation to the theory behind the experiment [31]. In our case, it is not very clear which metrics show that one algorithm is better than another. To mitigate this threat, we focused on the acceptance rate, novelty, and usefulness of the recommended commands, which are the most common metrics in the literature.

6 RESULTS

In this section, we report the results of our online study. We collected data about the students' interaction with the IDE during the entire school year and we detected 148 different user identifiers. Altogether, we detected 697,485 executions of 282 distinct commands. An average user identifier is associated with 4,713 executions of 32 distinct commands.

The distribution of the command executions and user's activity has a long tail. For instance, our data set contains 82 user identifiers with the interaction history shorter than 5 weeks and 77 identifiers that are associated with less than 500 command executions. We suspect that many of them belong to the same students, because we were not able to recognize and merge the additional identifiers without eliminating the anonymity or requiring some manual effort from the users in case they used different workspaces on different machines. However, because our experiments included only the users with long interaction histories, these additional identifiers do not affect the validity of our results. Nevertheless, we note that in a setting where this issue would be handled differently, the algorithms could produce different recommendations.

Our data set includes 28 users who started collecting interaction events on the first week of the study and were collecting them also after the RS GUI was available for installation. They were included in the experiment related to RQ1. 13 of these users installed the GUI and 15 did not, i.e., they also did not receive any recommendations and they served as a control group. In the experiment related to RQ2, we considered all users who installed the GUI, hence, since 6 additional GUI users started collecting data only after the first week of the study, the total number of users that we analysed is 19. These 19 users were also invited to answer the questionnaire related to RQ3, but only 13 responded.

To answer RQ1, we studied the effect of our IDE command RS on command discovery rates. Before the GUI was introduced, the 13 experiment participants who subsequently used it discovered 777 commands and the other 15 discovered 778 commands. After the GUI was made available, the 13 participants who used it discovered 288 commands, while the 15 participants who did not use the GUI discovered 103 commands. According to Pearson's χ^2 test, the difference between the number of discovered commands in the user samples is statistically significant ($p << 0.05$ and power $>> 0.95$). The effect size is above 0.34.

For a better illustration of the RS effect, we calculated the average number of known commands per week and the proportions of all known commands as they were discovered during the course of the study by the two user groups. We first counted for all individual users how many commands they used during the study and we computed how many distinct commands they executed until the end of every week, since the beginning of the study. Then we also calculated the proportions of known commands per week for each user and we calculated the average proportions for the two sets of users. Using these data, we plotted the learning curves, which are visualized in Fig. 2.

As can be seen in Fig. 2, already by the end of the week 16, an average study participant from the group of 15 students who did not use the RS executed more than 86% of the full

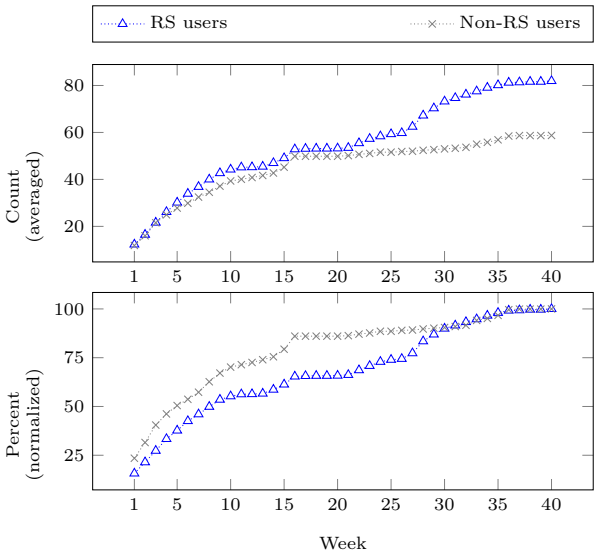

Figure 2: Command knowledge for two groups of users.

set of the commands that will be eventually used. When the RS was introduced, after week 26, the students in this group executed more than 89% of all the commands that they used in the study, while an average user of the RS GUI, from the group of 13 students, executed less than 75% of the total number of commands that will be eventually used.

We stress that among 288 commands discovered after the week 26 by the 13 participants who installed the RS GUI, only 58 were actually recommended. This shows that these students explored the IDE functionality autonomously and their command discovery rate increased considerably after the RS GUI introduction, compared to the previous weeks and compared to those who did not receive recommendations.

To answer RQ2, we studied how well accepted, novel, and useful are the commands recommended by the different algorithms. When we observe the measured recommendation acceptance rates, which are presented in Tab. 2, we see that the highest acceptance was achieved by the Most Popular algorithm. Amongst the 93 accepted recommendations, 50 were generated by this algorithm, which means that an average experiment participant later executed 2.6 commands recommended by it. CoDis recommended 26 commands that were later executed, which is 1.4 per user. And CNTX recommended 17 such commands, which is 0.9 per user. There is 31% probability that a command visualized in an IDE will be later executed, if it is recommended by Most Popular. If it is recommended by CoDis, the probability is 15%. And if it is recommended by CNTX, the probability is 10%. Most Popular had the highest acceptance rate for 11 participants, CoDis for 5, and CNTX for 2. One participant did not execute any recommended command.

Mann-Whitney's tests show that the difference between Most Popular and CNTX, according to the acceptance rates per user, is statistically significant ($p < 0.041$), but the power of the test is below 0.88, which makes the results unreliable. Conversely, the p value for CNTX and CoDis is above 0.28 and for CoDis and Most Popular it is above 0.14, hence, the differences are not statistically significant.

Table 2: Performance of algorithms in respect to acceptance of visualized recommendations.

Algorithm	Visualised recom.	Accepted recom.	Acceptance rate	Best for users
Most P.	160	50	31%	61%
CoDis	170	26	15%	28%
CNTX	178	17	10%	11%

When we asked how familiar was the recommendation recipient with the command before it was recommended, we received 290 answers: 100 correspond to the commands recommended by the Most Popular algorithm, 101 to the commands recommended by CoDis, and 89 to the commands recommended by CNTX. The most novel recommendations are provided by CNTX. In Fig. 3, which shows the answers to the question, depending on the algorithms that generated the recommendations, we can see that three quarters of the CNTX recommendations were completely new to the recipients. The same was the case for one half of the Most Popular recommendations and two thirds of the recommendations generated by CoDis. Pearson's χ^2 tests show that only the difference between CNTX and Most Popular, according to the novelty of the recommended commands, is statistically significant ($p < 0.0014$ and power $>> 0.95$); the effect size is above 0.81. The p value for CNTX and CoDis is above 0.43 and for CoDis and Most Popular it is above 0.06.

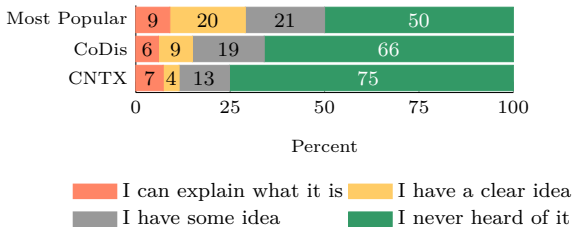

Figure 3: Familiarity with the recommended commands.

Furthermore, as can be seen in Fig. 4, according to the opinions of the two experts who assessed the commands' usefulness, all three algorithms mainly recommend useful commands, however, the degrees of usefulness vary between them. The average usefulness score for CNTX is 0.71, for CoDis it is 0.68, and for the Most Popular algorithm it is 1.36. According to Mann-Whitney's test, the differences between Most Popular and other two algorithms are statistically significant ($p << 0.05$ and power $>> 0.95$), while the difference between CNTX and CoDis is not ($p > 0.73$). The effect sizes for Most Popular in comparison with CNTX and CoDis are above 0.82 and 0.81, respectively. We note that it is not surprising that the most frequently executed commands are considered also the most useful ones: these are likely the commands that all users have to know and will eventually use. But, our results show that the other two algorithms mainly recommend useful commands as well.

Finally, to answer RQ3, we asked the recommendation recipients to tell us what they think about the received

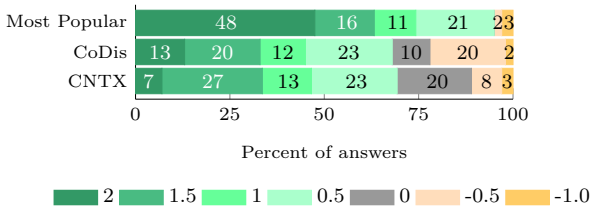

Figure 4: Usefulness scores of visualized commands.

recommendations. All the respondents liked the commands recommended by the system, i.e., 100% selected "Agree" for the first statement. 54% agree that the recommendations were relevant for their tasks, while others are undecided. 8% of the participants think that the system recommended too many bad commands, 15% are undecided, 69% disagree with the third statement, and 8% strongly disagree. These results show that the users have a generally positive opinion about recommendations, despite the fact that the overall acceptance rates were relatively low.

7 DISCUSSION

The results of the online evaluation confirm that the RS is well accepted by the users and that it serves its purpose, which is to increase the breadth of the used functionality provided by the IDE. With the experiment we showed that the number of newly discovered commands increases significantly, when the RS users start receiving recommendations. Hence, IDE command RSs are useful for learning new IDE functionality.

Additionally, we noticed that the command discovery rate increased much more than one would expect based on the recommendation acceptance rate. This finding is consistent with the observations of Li et al. [15], who introduced a command RS in another high-functionality application. It seems that beside promoting the usage of explicitly recommended commands, the RSs also promote exploration of the functionality in general, which is a beneficial side effect.

High acceptance rate achieved by the Most Popular algorithm indicates that the more coarse aggregation approach performs better on a small and uniform set of users, as in our experiment. We note that this result is also partially consistent with the results of the only previous online study, performed by Murphy-Hill et al. [19]. In general, neighbourhood-based and collaborative filtering algorithms, such as CNTX and CoDis, perform well when it is possible to identify groups of users with similar *tastes* [23], which are in our case represented with contexts and sets of known commands. However, if the entire user population represents only one relatively consistent group, as it was the case in our experiment, then this should not be the preferred approach.

Furthermore, we discovered that the users consider the commands recommended by the CNTX algorithm as the most novel. This means that our evaluation confirmed Gasparic et al.'s [5] hypothesis that low scores obtained by an algorithm according to standard offline evaluation metrics, such as recall, precision, and k-tail, indicate higher novelty of recommendations in a real-world setting. While a high novelty of recommendations can be achieved also by providing irrelevant recommendations, we can confirm that this was not the case for any of the evaluated algorithms. In fact, the average usefulness scores given to the recommended commands by two experts were positive for all three algorithms.

It seems that the list of popular Eclipse commands that are useful for all the users is much longer than we expected and even the users who know many commands may benefit from a RS that is based on a very simple algorithm, such as the Most Popular algorithm, for example. Consequently, we conjecture that the introduction of an IDE command RS during programming courses would be relatively simple and could significantly improve the IDE command knowledge of the students. However, considering the difference between accurate *predictions* and useful *recommendations*, which was discussed at the beginning of the paper, it may be that in a setting with more diverse and highly-experienced users who care for novel recommendations other algorithms would perform better. Because the IDE command recommendation task is different from the one addressed by traditional RSs, particularly from the perspective of the catalogue size, which is in our case much smaller than, for instance, in a typical movie RS, we want to encourage the community to continue exploring domain-specific techniques.

8 CONCLUSIONS

We have presented an IDE command RS and its evaluation. We integrated data collection tools, three different algorithms, and a GUI proposed in the literature. We deployed the complete RS and invited first year computer science students to participate in our one school year long study.

While previous work was focused on offline evaluations of recommendation algorithms, we addressed a set of research questions that are focused on the overall acceptance of the system in real-life settings. From the perspective of Siegmund et al.'s [26] considerations on internal and external validity in empirical software engineering, we consider our work as a necessary complement to the previous research.

The results of our study show that recommending the most popular (not yet used) commands to students can already increase the breadth of the used functionality and motivate them to learn more commands, even if the students already know a large set of commands. On the other hand, those who want to receive highly novel and also useful recommendations can use more advanced and more recent algorithms.

Knowing that an IDE command RS can improve the knowledge of the available IDE functionality, we think that it would be beneficial for the software engineering community to assess the actual effect of this additional knowledge on the productivity of software developers. While it is generally accepted that software development tools influence the efficiency and quality of software construction [1, 12, 25, 27, 30], it would be valuable to conduct quantitative studies to find out how much time is needed to learn more commands and how much time is saved by using them, for example. This type of research, which is out of the scope of this paper, would provide additional motivation for exploring IDE command RSs, particularly because the same community already developed a large set of additional IDE tools [6, 24], while in practice, an average IDE user uses only a very small subset of the functionality that is available in standard IDE versions [18].

REFERENCES

[1] A. Ahmed. 2011. *Software Project Management: A Process-Driven Approach*. Taylor & Francis.

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

[3] D. Campbell and M. Miller. 2008. Designing Refactoring Tools for Developers. In *Workshop on Refactoring Tools*.

[4] G. Fischer. 2001. User Modeling in Human–Computer Interaction. *User Modeling and User-Adapted Interaction* 11 (2001), 65–86.

[5] M. Gasparic, T. Gurbanov, and F. Ricci. 2017. Context-Aware Integrated Development Environment Command Recommender Systems. In *IEEE/ACM International Conference on Automated Software Engineering*.

[6] M. Gasparic and A. Janes. 2016. What recommendation systems for software engineering recommend: A systematic literature review. *Journal of Systems and Software* 113 (2016), 101–113.

[7] M. Gasparic, A. Janes, F. Ricci, G. C. Murphy, and T. Gurbanov. 2017. A graphical user interface for presenting integrated development environment command recommendations: Design, evaluation, and implementation. *Information and Software Technology* 92 (2017), 236–255.

[8] M. Gasparic, A. Janes, F. Ricci, and M. Zanellati. 2017. GUI Design for IDE Command Recommendations. In *International Conference on Intelligent User Interfaces*.

[9] M. Gasparic, G. C. Murphy, and F. Ricci. 2017. A context model for IDE-based recommendation systems. *Journal of Systems and Software* 128 (2017), 200–219.

[10] M. Gasparic and F. Ricci. 2017. Should Context-Aware IDE Command Recommendations Always Be Presented In-Context or Not?. In *AWARE workshop*.

[11] T. Grossman, G. Fitzmaurice, and R. Attar. 2009. A Survey of Software Learnability: Metrics, Methodologies and Guidelines. In *SIGCHI Conference on Human Factors in Computing Systems*.

[12] IEEE Computer Society, P. Bourque, and R. E. Fairley. 2014. *Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society Press.

[13] M. Kersten and G. C. Murphy. 2006. Using Task Context to Improve Programmer Productivity. In *ACM/SIGSOFT International Symposium on Foundations of Software Engineering*.

[14] B. P. Knijnenburg, M. C. Willemsen, Z. Gantner, H. Soncu, and C. Newell. 2012. Explaining the User Experience of Recommender Systems. *User Modeling and User-Adapted Interaction* 22 (2012), 441–504.

[15] W. Li, J. Matejka, T. Grossman, J. A. Konstan, and G. Fitzmaurice. 2011. Design and Evaluation of a Command Recommendation System for Software Applications. *ACM Transactions on Computer-Human Interaction* 18 (2011), 6:1–6:35.

[16] J. Matejka, W. Li, T. Grossman, and G. Fitzmaurice. 2009. CommunityCommands: Command Recommendations for Software Applications. In *ACM Symposium on User Interface Software and Technology*.

[17] S. M. McNee, J. Riedl, and J. A. Konstan. 2006. Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems. In *CHI Extended Abstracts on Human Factors in Computing Systems*.

[18] E. Murphy-Hill. 2012. Continuous Social Screencasting to Facilitate Software Tool Discovery. In *International Conference on Software Engineering*.

[19] E. Murphy-Hill, R. Jiresal, and G. C. Murphy. 2012. Improving Software Developers' Fluency by Recommending Development Environment Commands. In *ACM/SIGSOFT International Symposium on the Foundations of Software Engineering*.

[20] E. Murphy-Hill, D. Y. Lee, G. C. Murphy, and J. McGrenere. 2015. How Do Users Discover New Tools in Software Development and Beyond? *Computer Supported Cooperative Work* 24 (2015), 389–422.

[21] P. Resnick and H. R. Varian. 1997. Recommender Systems. *Commun. ACM* 40 (1997), 56–58.

[22] F. Ricci. 2014. Recommender Systems: Models and Techniques. In *Encyclopedia of Social Network Analysis and Mining*. Springer, 1511–1522.

[23] F. Ricci, L. Rokach, and B. Shapira. 2015. Recommender Systems: Introduction and Challenges. In *Recommender Systems Handbook*. Springer, 1–34.

[24] M. P. Robillard and R. J. Walker. 2014. An Introduction to Recommendation Systems in Software Engineering. In *Recommendation Systems in Software Engineering*. Springer, 1–11.

[25] W. Scacchi. 1991. Understanding software productivity: towards a Knowledge-based approach. *International Journal of Software Engineering and Knowledge Engineering* 1 (1991), 293–321.

[26] Janet Siegmund, Norbert Siegmund, and Sven Apel. 2015. Views on Internal and External Validity in Empirical Software Engineering. In *International Conference on Software Engineering*.

[27] I. Sommerville. 2011. *Software Engineering*. Pearson.

[28] S. Stuckemann. 2015. Vignelli - Automated Design Guidance for Developers. *Imperial College London* (2015).

[29] P. Viriyakattiyaporn and G. C. Murphy. 2010. Improving Program Navigation with an Active Help System. In *Conference of the Center for Advanced Studies on Collaborative Research*.

[30] B. Walraet. 2014. *A Discipline of Software Engineering*. Elsevier.

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

[32] S. Zolaktaf and G. C. Murphy. 2015. What to Learn Next: Recommending Commands in a Feature-Rich Environment. In *International Conference on Machine Learning and Applications*.