Design and Evaluation of a Command Recommendation System for Software Applications

WEI LI, JUSTIN MATEJKA, and TOVI GROSSMAN, Autodesk Research
JOSEPH A. KONSTAN, University of Minnesota
GEORGE FITZMAURICE, Autodesk Research

We examine the use of modern recommender system technology to aid command awareness in complex software applications. We first describe our adaptation of traditional recommender system algorithms to meet the unique requirements presented by the domain of software commands. A user study showed that our item-based collaborative filtering algorithm generates 2.1 times as many good suggestions as existing techniques. Motivated by these positive results, we propose a design space framework and its associated algorithms to support both global and contextual recommendations. To evaluate the algorithms, we developed the CommunityCommands plug-in for AutoCAD. This plug-in enabled us to perform a 6-week user study of real-time, within-application command recommendations in actual working environments. We report and visualize command usage behaviors during the study, and discuss how the recommendations affected users behaviors. In particular, we found that the plug-in successfully exposed users to new commands, as unique commands issued significantly increased.

Categories and Subject Descriptors: H.2.8 [**Database Management**]: Database Applications—*Data mining*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Graphical user interfaces*

General Terms: Design, Human Factors, Algorithms

Additional Key Words and Phrases: Collaborative filtering, command recommender system

ACM Reference Format:
Li, W., Matejka, J., Grossman, T., Konstan, J. A., and Fitzmaurice, G. 2011. Design and evaluation of a command recommendation system for software applications. ACM Trans. Comput.-Hum. Interact. 18, 2, Article 6 (June 2011), 35 pages.
DOI = 10.1145/1970378.1970380 http://doi.acm.org/10.1145/1970378.1970380

1. INTRODUCTION

Many of today's programs have not hundreds, but thousands of commands for a user to become aware of and learn [Hsi and Potts 2000]. In each release, more commands might be added to provide new functionality, and without explicit effort on the part of the user to learn about the new features, they are left untouched [Baecker et al. 2000]. For example, in Autodesk's AutoCAD, the number of commands has being growing linearly over time (Figure 1) with 45 new commands, all providing new features, added in the most recent 2011 release. And even with the hundreds of commands available in AutoCAD, 90% of users use less than 90 commands (Figure 2).

An inherent challenge with such systems is a user's awareness of the functionality which is relevant to their specific goals and needs [Findlater and McGrenere 2010;

Section 4 of this article is based on Matejka et al. [2009].
Authors' addresses: W. Li, J. Matejka, T. Grossman, and G. Fitzmaurice, Autodesk Research, Toronto, Canada; J. A. Konstan, Grouplens Research, University of Minnesota, Minneapolis, MN; email: wei.li@autodesk.com.

DOI 10.1145/1970378.1970380 http://doi.acm.org/10.1145/1970378.1970380

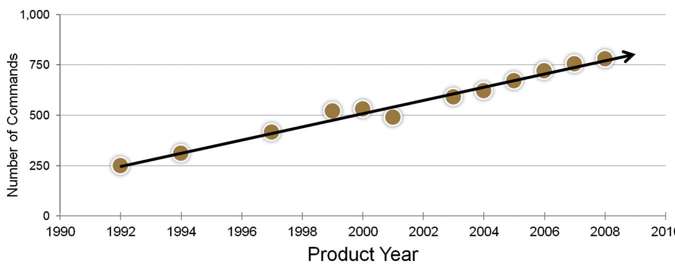

Fig. 1. Number of built-in commands in each AutoCAD yearly release.

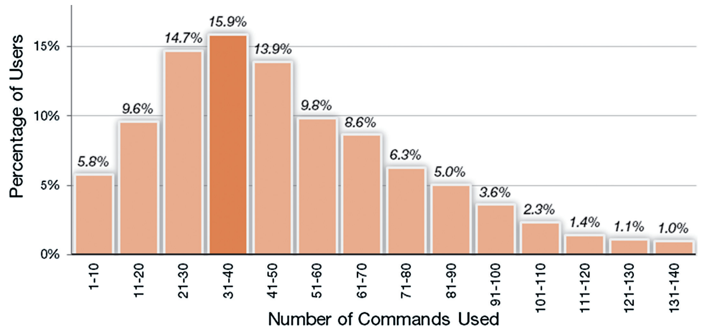

Fig. 2. Histogram of the number of commands used by 4000 AutoCAD users in 6 months. The largest group of users use only between 31 and 40 commands.

Grossman et al. 2009; Shneiderman 1983]. Awareness of functionality is important not only for learning how to accomplish new tasks, but also for learning how to better accomplish existing tasks. In a potential "best case scenario," the user works with an expert next to them, who can recommend commands when appropriate.

While previous HCI literature has looked at intelligent online agents, most of this work is focused on predicting what current state a user is in, if they require assistance, and how to overcome problems [Davison and Hirsh 1998; Dent et al. 1992; Hermens and Schlimmer 1993; Horvitz 1999; Igarashi and Hughes 2001; Krzywicki et al. 2010; Miller et al. 2003; Shneiderman 1983]. Existing software techniques for suggesting commands, such as "tip-of-day" and "did you know," can expose users to functionality, but are often presenting irrelevant information to the user [Fischer 2001; Norman and Draper 1986]. Linton's OWL system [Linton and Schaefer 2000] recommended commands to word processing users based on the common usage pattern of other users and the areas where the target user differed from those patterns. This personalization assumes that all users have similar optimal usage patterns and could be extended to take into account the user's current context of use. In addition, such working implementations of command recommenders embedded within target applications have never been evaluated during real usage situations. Thus, it is unknown how well such recommendations will be received by end users, and what the important usability issues are concerning their delivery.

Systems that recommend relevant content to users, known as "recommender systems" are very popular in other domains. Some of the most popular movie, shopping, and music websites provide users with personalized recommendations [Linden et al. 2003; McNee et al. 2006; Resnick et al. 1994; Sarwar et al. 2000], and research in improving the *collaborative filtering* algorithms that these systems use is an active field of research [Adomavicius and Tuzhilin 2005]. This includes work which use contextual or short-term temporal information to improve recommendations [Nguyen and Ricci 2008; Schafer et al. 1999; Terveen et al. 2002].

In this article we introduce and investigate the application of such modern collaborative filtering algorithms to address the command awareness problem in software applications. Our goal is to present to users within-application recommendations of commands they are not currently familiar with that will help them use the software more effectively. This article is divided into three main parts.

First, we describe the development and evaluation of collaborative filtering algorithms that have been specifically designed for the domain of software command recommendations. The findings from this study validated the usage of such algorithms, as they provided significantly improved recommendations in comparison to existing approaches.

Second, we systematically investigate the design space of these algorithms, and extend them to support the delivery of short-term contextual recommendations in addition to traditional "long-term" global recommendations. An offline evaluation indicated that this enhancement could further improve the recommendations.

Third, we describe our *CommunityCommands* system, developed as a plug-in for AutoCAD. The plug-in applies our developed recommender system algorithms to generate personalized command recommendations displayed in a peripheral tool palette within the user interface. Using this system, we conducted an online evaluation where we deployed the plug-in to 32 full-time professional AutoCAD users for a 6-week user study in their real working environments. We report and visualize the data collected during this online evaluation, including metrics of usage patterns and adoption rates, and discuss subjective feedback. Both the contextual and long-term algorithms successfully exposed users to new commands as there was a significantly increase in the number of unique commands issued, and a subjective user-preference towards contextual recommendations was reported.

1.1. Summary of Contributions

The work that follows in this article makes to following contributions:

1. We adapt modern collaborative filtering algorithms to the domain of software command recommendations and provide a thorough discussion of the issues which need to be considered, and possible variations.
2. We introduce a design space for such algorithms that considers methods for delivering both short-term (contextual) and long-term (global) recommendations.
3. We introduce a new offline evaluation algorithm for measuring the effectiveness of such algorithms from existing usage data logs.
4. We perform an online evaluation of our new algorithms and show that they significantly improve the quality of recommendations compared to existing approaches.
5. We describe a new user interface design for delivering and updating command recommendations in real time, within the actual application.
6. We deploy and evaluate our new interface component as a plug-in for AutoCAD in a field study with 32 users.
7. We provide a thorough analysis of the results from our field study, which showed that there was a significant increase in the number of unique commands issued when our plug-in was activated.

2. RELATED WORK

To our knowledge, CommunityCommands is the first recommender system to use collaborative filtering technology to address the problem of learning software commands.

However, our work is related to a number of previous research topics, both in the areas of software usability and recommender systems.

2.1. Adaptive and Intelligent User Interfaces

Previous research has focused on adapting and optimizing the interface to the current user and context [Claypool et al. 2001; Cutrell et al. 2000; Gajos et al. 2010], and inferring possible operations based on the user's behavior [Horvitz et al. 1998; Hudson et al. 2003]. The domain knowledge base of such systems is often predesigned and self-contained. With CommunityCommands, the knowledge base is acquired automatically from a large scale user community and evolves with its users.

Intelligent user interfaces are designed to observe and learn from users' actions, and accomplish personalized tasks. Examples include predicting the user's next command [Davison and Hirsh 1998], automatically completing forms [Hermens and Schlimmer 1993], maintaining calendars and emails [Dent et al. 1992; Horvitz 1999; McCrickard et al. 2003], assisting users in word processing tasks [Horvitz et al. 1998; Liu 2003], and automatically generating user interfaces adapted to a person's abilities and preferences [Gajos et al. 2010]. Most personal assistance programs analyze repetitive user behaviors, and automate the repetitions; in contrast, our system suggests useful commands that users have never used.

2.2. Notifications and Interuptability

Previous research [Bailey and Konstan 2006; Cutrell et al. 2001; Cutrell et al. 2000; McGrenere et al. 2002] has demonstrated the harmful effects that notifications can have on a user's task performance. To compensate, there is a large body of work on determining when and how to best interrupt a user [Hudson et al. 2003]. However, this remains an open problem. CommunityCommands uses an ambient display [Wisneski et al. 1998], where the user can get the information when they are ready, thus avoiding the problems associated with interrupting the user's work flow.

2.3. Recommender Systems

Recommender systems have become an important approach to help users deal with information overload by providing personalized suggestions [Hill et al. 1995; Resnick et al. 1994; Sarwar et al. 2000], and have been successfully applied in both industry and academia. Recommender systems support users by identifying interesting products and services when the number and diversity of choices outstrips the user's ability to make wise decisions. One of the most promising recommending technologies is collaborative filtering [Hill et al. 1995; Sarwar et al. 2000]. Essentially, a nearest-neighbor method is applied to find similar users and recommendations are based on how her likes and dislikes relate to a large user community. Examples of such applications include recommending movies [Miller et al. 2003], news [Resnick et al. 1994], books [Linden et al. 2003; Sarwar et al. 2000], music [AppleComputers], research papers [McNee et al. 2006a], and school courses [Farzan and Brusilovsky 2006; Hsu 2008]. However, research has shown that users may be reluctant to provide explicit ratings [Claypool et al. 2001], and so our research considers an implicit rating system for software commands.

2.4. Recommending Commands

Under the assumption that more functionality makes a system more useful and marketable [Baecker et al. 2000] applications have continued to grow in both commands and features, with some of the more complex systems containing thousands of commands. This has resulted in software "bloat" [Baecker et al. 2000; Kaufman and Weed 1998] which can overwhelm users and make it difficult to become aware of which features are relevant to their personal needs [Baecker et al. 2000; Findlater and McGrenere 2010]. While efforts have been made to reduce the feature space of applications through adaptive [Findlater and McGrenere 2010; Shneiderman 2003] or multilayered [Carroll and Carrithers 1984; Findlater and McGrenere 2010; Shneiderman 2003] user interfaces, our focus is instead on a command recommendation system which helps users find relevant tools that they were not previously aware of. Little research has been conducted to help users learn and explore a complicated software package using a recommender system. Typical approaches to proactively introduce functionality to a user include "Tip of the day," and "Did you know" [Norman and Draper 1986], but these suggestions are often irrelevant to the user and are presented in a decontextualized way [Fischer 2001].

The OWL System [Linton and Schaefer 2000] is one of the few systems we have identified as going beyond these simple solutions. The system, which is meant to run within an organization, compares a user's command usage to the average usages across the organization. The system then makes a recommendation if a command is being underutilized or overutilized by an individual in comparison to the group. This algorithm produces recommendations based on the assumption that all users in the organization should share the same command usage distribution. Accordingly, it may work well for novice users engaged in common tasks—for instance, by informing a word processing user that there is a delete word operation. The underlying assumption, that users should exhibit similar application usage patterns, may break down as the users become more specialized and have different tasks, goals, and preferences. The system which Linton described is a useful first step, but could potentially be improved when combined with more advanced recommender system technology. CommunityCommands uses collaborative filtering to recommend the most relevant commands for each individual user.

2.5. Recommendations Based on Long-Term and Session-Specific Preference

In addition to being personalized, recommendations could also be contextual. Schafer et al. [1999] introduced the concept of degree of persistence in recommendations, which depends on how the recommendations are based on data from previous customer sessions with the system. This dimension ranges from completely ephemeral recommendations, which are based on a single customer session, to persistent recommendations, which are based on the system recognizing the customer and exploiting information previously gathered about her over a longer period of time.

The length of a user's rating history is an important factor to consider when creating a recommendation system. If the system includes the ability to track users across multiple sessions then the set of recommendations can be based on a long-term understanding of the user's preferences, for example, the movie recommender in Netflix.[1] On the other hand, if profiles are derived from anonymous user sessions, then the recommendations will be based on a short-term view of each user's history, such as in the CDNow[2] music recommendation system. Nguyen and Ricci [2008] performed offline evaluations showing prediction accuracy can be improved by combining both long-term and session-specific preference. In our work we investigate the design space of short-term and long-term command recommendations.

2.6. Recommender System Evaluations

Research on recommender systems is generally focused on improving their prediction accuracy [Herlocker et al. 2004]. To do this, offline evaluation divides the user dataset into a training dataset and a testing dataset. Some part of the user rating data in the testing set is removed and those values are predicted using the training data and the uncovered parts of the testing data [Basu et al. 1998; Herlocker et al. 2002; Herlocker et al. 1999; Sarwar et al. 2000]. Offline evaluation can be an efficient way to evaluate

[1]www.Netflix.com
[2]www.CDNow.com

recommenders when looking at accuracy metrics. However, accuracy metrics are not able to evaluate the novelty or serendipity of recommendations [Herlocker et al. 2004; McNee et al. 2006b]. Although much effort has gone into offline evaluation of recommendation novelty [Celma and Herrera 2008; McNee et al. 2002; Weng et al. 2007; Zhang et al. 2002], online evaluation, with real user participation, is still the most accurate way to evaluate the novelty of recommendations [McNee et al. 2006b] and the only way to evaluate actual recommendation usefulness, user confidence and decision effort [Chen and Pu 2006]. To evaluate the strength of our proposed recommendation algorithms, we use both online and offline evaluations [Matejka et al. 2009] and consider both novelty and usefulness. We also study the change of overall new command adoption rate with and without the recommenders.

2.7. New Command Adoption

The issue of experienced users learning new things within a software application is a difficult problem [Bosser 1987; Linton and Schaefer 2000; Mack 1990]. Bosser [1987] noted that people stop learning new things quickly. Mack [1990] observed that people tend to stay with their established methods. Our hope is to design new algorithms to generate useful recommendations, along with a simple and unobtrusive interface to deliver those recommendations, to best encourage new feature learning and adoption.

3. SYSTEM OVERVIEW AND APPLICATION DOMAIN

Our work is implemented within AutoCAD, a widely used architecture and design software application, made by Autodesk. We felt AutoCAD would be an excellent software package to work with, since it not only has thousands of commands (including both built-in commands, system variables, and customized commands), but also numerous domains of usage. While our work is implemented within AutoCAD, the concepts map to any software where command awareness may be an issue, and its usage varies across users.

In AutoCAD, command usage histories are collected using a Customer Involvement Program (CIP), which is a voluntary program that individual users can choose to opt into. From this program, we obtained our data set, which is composed of 40 million (User, Command, Time) tuples collected from 16,000 AutoCAD users over a period of 6 months. The CIP data does not contain any personal information, specific command parameters, or document content, so a user's confidentiality is respected.

The general idea for the CommunityCommands system is to first compare a target user's command usage to the entire user population. The system then generates personalized recommendations of commands to the target user, and presents a top-N list of those recommendations within a palette in the application (Figure 3). In Sections 4 and 5 we focus on the algorithms used to generate the command recommendations. In Section 6, we describe the design interface used to present those recommendations to the user, and a study where we deployed the interface to real AutoCAD users to provide in-application, real-time, command recommendations.

4. COLLABORATIVE FILTERING FOR COMMAND RECOMMENDATIONS[3]

As alluded to in our review of related work, there are a number of unique considerations to address in developing a recommender system for software commands. Traditional

[3]The contents of Section 4 are based on the existing ACM UIST publication:

Matejka, J., Li, W., Grossman, T., and Fitzmaurice, G. 2009. CommunityCommands: command recommendations for software applications. In Proceedings of the 22nd Annual ACM Symposium on User interface Software and Technology (Victoria, BC, Canada, October 04–07, 2009). UIST'09. ACM, New York, NY, 193–202. DOI= http://doi.acm.org/10.1145/1622176.1622214.

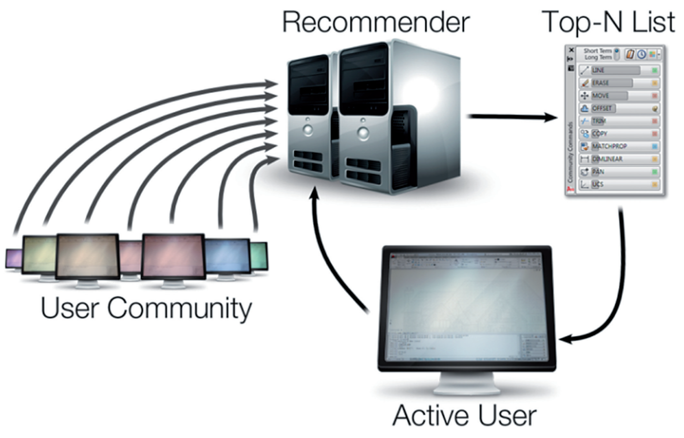

Fig. 3. CommunityCommands system overview.

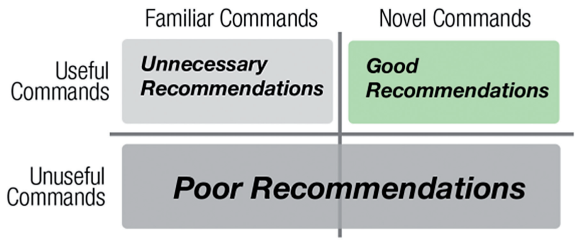

Fig. 4. Map of Good, Poor, and Unnecessary Recommendations.

approaches to recommender system development must be rethought, such as how ratings are defined and how collaborative filtering algorithms are chosen. In the following section, we describe these considerations in more detail.

4.1. Algorithm Design Considerations

Our goal is to develop algorithms that will deliver recommendations that are both novel and useful.

1. *Novel Recommendations.* The recommendations should be commands that the user is unfamiliar with. This is in contrast to Linton's work, where recommendations were also made to increase or decrease existing command usages [Linton and Schaefer 2000].
2. *Useful Recommendations.* The recommended commands also need to be useful for the user. This could mean that the command is useful immediately, or useful at some point in the future given the type of work the user does.

The combination of novel and useful commands leads to a two-dimensional space of command recommendation types (Figure 4).

We consider a good recommendation to be a command that is both useful and novel to the user. A poor recommendation is a command that is not useful to the user. An unnecessary recommendation is a command which is useful to the user, but the user was already familiar with. Unnecessary recommendations can actually be helpful in improving the user's confidence in the system [McNee et al. 2006a], but this is very dependent on the user's expectations. If the expectation is that the system will be

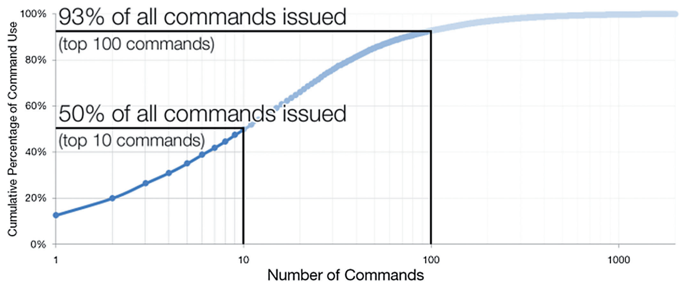

Fig. 5. Cumulative percentage of command counts for AutoCAD commands.

suggesting "new" commands which may be useful, commands with which the user is already familiar may be seen as poor suggestions.

4.2. The "Ratings"

Typical recommender systems depend on a rating system for the items which it recommends. For example, a recommender system for movies may base its recommendations on the number of stars that users have assigned to various titles. These ratings can be used to find similar users, identify similar items, and ultimately, make recommendations based on what it predicts would be highly rated by a user.

Unfortunately, in our domain, no such explicit rating system exists. Instead, we implicitly base a user's "rating" for any command on the frequency for which that command is used. Our collaborative filtering algorithm then predicts how the user would "rate" the commands which they do not use. In other words, we take a user's *observed* command-frequency table as input, and produce an *expected* command-frequency table as output.

A basic method is to define the rating r_{ij} of a command c_i as the frequency for which the user u_j has used the command c_i. A limitation of using this approach is that in general, a small number of commands will be frequently used by almost everyone [Lafreniere et al. 2010; Zipf 1949]. Thus, when comparing users, each pair of users will tend to have high similarity because they will all share these popular high frequency commands. This is certainly the case in AutoCAD. For example, in Figure 5, we see that the top 10 commands make up 50% of all commands issued, and the top 100 commands make up 93% of all commands issued.

We need to suppress the overriding influence of commands that are being used frequently and by many users. Document retrieval algorithms face a similar challenge. For example, an Internet search engine should not consider two Webpages similar because they both share high frequencies of the words "a," "to," and "the." Such systems use a "term frequency inverse document frequency" *(tf-idf)* technique [Jones 1972] to determine how important a word is to a particular document in a collection. For our purposes, we adapt this technique into a *command frequency, inverse user frequency (cf–iuf)* weighting function, by considering how important a command is to a particular user within a community. To do so, we first take the command frequency *(cf)* to give a measure of the importance of the command c_i to the particular user u_j.

$$cf_{ij} = \frac{n_{ij}}{\sum_k n_{kj}},$$

where n_{ij} is the number of occurrences of the considered command of user u_j, and the denominator is the number of occurrences of all commands of user u_j.

The inverse user frequency *(iuf)*, a measure of the general importance of the command, is based on the percentage of total users that use it:

$$iuf_i = log\frac{|S|}{|\{u_j : c_i \in u_j\}|},$$

where:

$|S|$: total number of users in the community
$|\{u_j : c_i \in u_j\}|$: number of users who use c_i.

With those two metrics we can compute the *cf-iuf* as:

$$cf\text{-}iuf_{ij} = cf_{ij} \cdot iuf_{ij}.$$

A high weight in $cf\text{-}iuf$ is obtained when a command is used frequently by a particular user, but is used by a relatively small portion of the overall population.

4.3. Generating Recommendations

Collaborative filtering approaches are classified as model-based or memory-based. Model-based approaches use training data to generate a model which is able to predict the ratings for unrated items [Breese et al. 1998; Canny 2002; Hofmann 2004]. However, model-based approaches often need to tune a large number of parameters and therefore they are hard to apply in practical systems. Memory-based approaches generate recommendations based on all ratings stored into memory without learning an abstract model. The advantages of the memory-based methods include: 1) fewer parameters have to be tuned; and 2) their recommendations are more intuitive for users to understand. The disadvantage of a memory-based approach is that data scarcity stops many memory-based methods from being practical. But in our application, a user's command data is usually less sparse as compared to data in current recommender systems such as Netflix and Amazon. For example there are more than 8 million items on Amazon's Website compared to only two thousand commands in AutoCAD.

Based on above considerations, we choose to focus on memory-based collaborative filtering techniques. The two most common such techniques are user-based [Resnick et al. 1994] and item-based [Sarwar et al. 2001] algorithms, but it is not clear from prior literature which of these two approaches will be best suited for our specific domain. As such, we will investigate both algorithms in this section.

Both of these techniques have two inputs: the command history of each user in the community, and the command history of the user for whom we are generating a recommendation, called the active user. User-based collaborative filtering predicts an active user's interest in an item by looking at rating information from similar user profiles. Item-based approaches apply the same idea, but use similarity between items instead of users. User-based algorithms need to search for neighbors among a large user population of potential neighbors. Item-based algorithms avoid this step by exploring the relationships between items first. Recommendations are computed by finding items that are similar to other items the active user has liked. Because the relationships between items are relatively static, item-based algorithms typically provide the same quality as the user-based algorithms with less online computation time [Sarwar et al. 2001].

4.3.1. User-Based Collaborative Filtering. User-based collaborative filtering generates recommendations for an active user based on the group of individuals from the community that he/she is most similar to (Figure 6). The algorithm averages this group's command frequencies, to generate an expected command-frequency table for the active user. The algorithm details are described in the following.

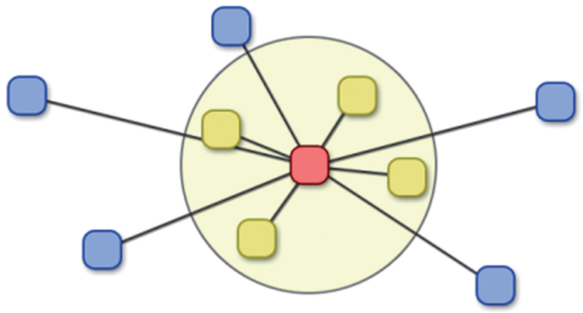

Fig. 6. The active user is in red, and her expected frequency table will be compiled from her most similar neighbors (in yellow).

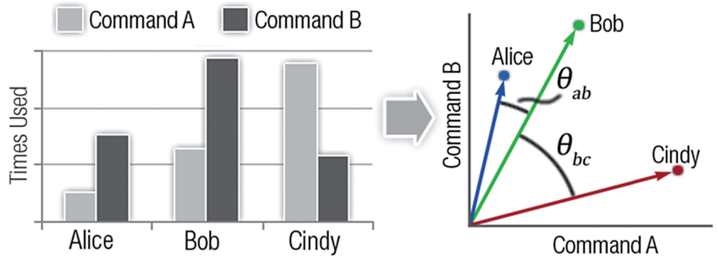

Fig. 7. Simplified example of user similarity. Alice and Bob are more similar than Bob and Cindy as the angle between their command vectors is smaller.

Defining Command Vectors. For user-based collaborative filtering we require a method to measure the similarity between two users. A common approach for doing this is to first define a representative vector for each user, and then compare the vectors. Our method is to define the command vector V_j such that each cell, $V_j(i)$, contains the *cf-iuf* value for each command c_i, and use these vectors to compute user similarity.

Finding Similar Users. As with many traditional recommender systems, we measure the similarity between users by calculating the cosine of the angle between the users' vectors [Witten et al. 1984]. In our case, we use the command vectors, as described above. Considering two users u_A and u_B with command vectors V_A and V_B

$$\text{similarity}\,(u_A, u_B) = \cos\,(\theta_{V_A,V_B}) = \frac{V_A \cdot V_B}{\| V_A \| * \| V_B \|}.$$

Thus, when *similarity* is near 0, the vectors V_A and V_B are substantially orthogonal (and the users are determined to be not very similar) and when *similarity* is close to 1 they are nearly collinear (and the users are then determined to be quite similar). As can be seen in Figure 7, using the cosine works nicely with our rating system based on frequencies, since it does not take into account the total number of times a user has used a command, but only its relative frequency. We compare the active user to all other users in the community, to find the n most similar users, where n is another tuning parameter.

Calculating Expected Frequencies. To calculate an expected frequency for each command, we take a weighted average of the command frequencies for the active user's n similar users. We used offline evaluations to tune n, size of neighbors, eventually converging on a value of $n = 200$. We define the expected frequency, ef_{ij}, for command

c_i and user u_j:

$$ef_{ij} = \sum_{k=1}^{n} w_{jk} cf_{ik},$$

where $w_{ij} = similarity(u_i, u_j)$ is and cf_{ik} is the frequency of command c_i and user k.

Removing Previously Used Commands. Once we create a list of all the command frequencies, we remove any command which the user has been observed to use, preventing known commands from being suggested.

Returning the Top-N List. The final step is to sort the remaining commands by their expected frequencies. The highest *N* commands will appear in the user's recommendation list.

4.3.2. Item-Based Collaborative Filtering. Rather than matching users based on their command usage, our item-based collaborative filtering algorithm matches the active user's commands to similar commands. The steps of the algorithms are described below.

Defining User Vectors. We first define a vector V_i for each command c_i in the n dimensional user-space. Similar to user-based approach, each cell, $V_i(j)$, contains the *cf-iuf* value for each user u_j.

Build a Command-to-Command Similarity Matrix. Next, we generate a command-to-command similarity matrix, M. M_{ik} is defined for each pair of commands i and k as:

$$M_{ik} = \cos(V_i, V_k).$$

Create an "Active List". For the active user, u_j, we create an "active list" L, which contains all of the commands that the active user has used.

$$L_j = \{c_i | cf_{ij} > 0\}.$$

Find Similar Unused Commands. Next, we define a similarity score, s_i, for each command c_i which is not in the active user's active list:

$$s_i = \text{average}(M_{ik}, \forall c_k \in L).$$

Generate Top-N List. The last step is to sort the unused commands by their similarity scores s_i, and to provide the top N commands in the user's recommendation list.

4.4. Domain-Specific Rules

These techniques work without any specific knowledge about the application. In an initial pilot study, this was shown to lead to some poor recommendations which could have been avoided. Thus, we created two types of rules to inject some basic domain knowledge into the system.

Upgrades ($A \nRightarrow B$). An upgrade is a situation where if you use command A there is no need for you to use command B. For example, if an AutoCAD user uses MouseWheelPan we would not recommend the Pan command, since it is a less efficient mechanism to activate the same function.

Equivalencies ($A \nLeftrightarrow B$). We consider two commands to be "equivalent" when it makes sense for a user to use one of the two commands, but not both. For example in AutoCAD there are the *HATCH* and *BHATCH* commands. *BHATCH* is from earlier versions of the product, but it does the same thing.

We generated domain specific rules by consulting with an AutoCAD domain expert in a single, two-hour session. To generate the rules, we first explained to the domain expert our goal and the two types of rules. The expert was then presented with a list of the most frequently recommended commands from the pilot study as well as the most

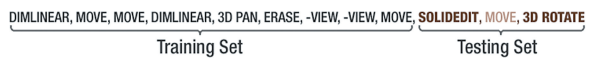
DIMLINEAR, MOVE, MOVE, DIMLINEAR, 3D PAN, ERASE, -VIEW, -VIEW, MOVE, SOLIDEDIT, MOVE, 3D ROTATE
Training Set
Testing Set

Fig. 8. k-Tail evaluation of a command sequence.

common commands from the CIP data. The expert inspected each command, one-at-a-time, and determined if there was another command already examined, or that he was aware of, for which one of the two rules applied. This process resulted in 21 specific rules being identified. Four of these rules were *Upgrades* and 17 of the rules were *Equivalencies*.

4.5. Offline Algorithm Evaluation

Here, we present an automated method to evaluate the recommender algorithms using our existing CIP data. Using this data set we can try to simulate the behavior of users that interact with a recommender system. Offline experiments are important because they require no interaction with real users, and thus allow us to compare a wide range of candidate algorithms at a low cost. Although offline evaluation cannot replace online evaluation, it is a useful step to tune the algorithms and verify our design decisions before the recommender system is deployed to real users.

The development of the algorithm was a challenging task since we required a metric that would indicate if a recommended command, which had never been observed, would be useful to a user. To accomplish this we developed a new *k-tail evaluation* where we use the first part of a user's command history as a training set, and the rest of the history as a testing set.

4.5.1. Evaluation Method: k-Tail. Consider a user u_i with a series of commands S. k-tail evaluation divides this command sequence into a training sequence S_{train} and a testing sequence S_{test} based on the number of unique commands, so that there are k unique commands in S_{test} which are not in S_{train}. For example, the command sequence in Figure 8 is a 2-tail series since there are two commands, SOLIDEDIT and 3D ROTATE, which have never appeared in the training set. To evaluate an algorithm, we find the average number of commands which are in both user i's recommendation list R_i, and their testing set $S_{test,i}$. We define a hit for a user, ($hit_i = 1$, otherwise 0), when $|R_i \cap ST_i| > 0$. $Recall_{kTail}$ returns the percentage of users for which a hit occurred:

$$Recall_{kTail} = \frac{\sum_1^n hit_i}{n},$$

where n is the number of users. Thus, $Recall_{kTail}$ returns the percentage of users whose top-N recommendation list includes at least one out of k commands in the testing set.

In addition to testing our user-based and item-based collaborative filtering algorithms, we also implemented and evaluated Linton's algorithm [Linton and Schaefer 2000]. The algorithm suggests the top commands, as averaged across the whole user population, that a user has not used.

4.5.2. Offline Results. All three recommendation algorithms were evaluated using the k-tail method and the offline CIP data. We only included users for which we had observed at least 2000 commands (4033 total users). We used this cutoff because when fewer commands have been logged, there is an increased chance that an unused command is familiar to the user and has simply not been recorded yet. The command sequence of each CIP user is divided into a training set and a k-tail.

Figure 9 shows that when $k = 1$, the item-based algorithm predicts the next new command correctly for 850 users, about 240 more than Linton's. $Recall_{kTail}$ returns the percentage of users whose top-N recommendation list includes at least one out of k commands in the testing set. When $k > 1$, it is more likely that at least one of those

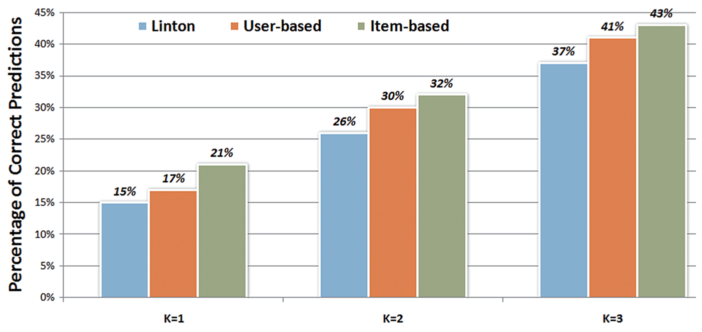

Fig. 9. Offline results showing $Recall_{kTail}$, the percentage of times the next new command was predicted in a list of 10 by each algorithm.

new commands in the testing set (tail) is recommended. In the rest of this paper we simply use $k = 1$, because it is the most difficult prediction based on our definition.

The downside of offline experiments is that they can answer a very narrow set of questions, typically questions about the prediction power of an algorithm. To evaluate this new collaborative filtering system in software learnability domain, a necessary follow-up is to perform an online user study.

4.6. Online Algorithm Evaluation

We performed offline experiments using existing data sets to estimate recommender performance measures such as prediction accuracy. While our offline evaluation showed promise for our new techniques, the results may not be fully indicative of how our algorithms would work in practice. As such, we conducted an online study with real users. We collected data for a set of real users, generated personalized recommendations for each of them, and had them evaluate their recommendations in a web-based survey.

4.6.1. Participants and Setup. We recruited 36 users (25 male, 11 female) of AutoCAD 2009 to participate in the study. To be considered for the study users were required to use AutoCAD a minimum of 20 hours per week. Participants were aged 21 to 55 and worked in varying fields including architecture, civil planning, and mechanical design, across North America.

To capture the participants' command usages, we instrumented their systems with a custom written application to give us access to their full CIP data from the time it was installed. Participants were asked to continue using AutoCAD as they normally would. Command data was recorded from each user for approximately 10 weeks. After collecting each user's data, the recommendations were generated.

4.6.2. Generating Recommendations. We used a within-participant design. That is, each participant was sent recommendations from each of the three algorithms. To do this, we generated a top 8 list for each of the three algorithms. We then took the union of these three lists, and randomized the order. Since the algorithms could produce common commands, the final lists could range in size, and in the study, ranged from 14 to 22 items.

Each user was sent a survey with questions about each of the commands in their customized recommendation list. For each recommended command participants were given a short description of the functionality (for example "XLINE: creates an infinite line"). Users were asked to familiarize themselves with the command as much as possible before answering any questions about it.

Participants were asked to rate the commands on the following 2 statements, using a 5-point Likert scale:

Q1. I was familiar with this command.
Q2. I will use this command.

In an initial pilot study, we found that users sometimes claimed to use a command frequently, when we could tell from their data that they did not. This was often due to two different commands sounding similar. As such, in this study, we made it clear to the participants that they had not used any of the listed commands.

During the course of the study we stopped receiving data from 12 of the participants (they changed computers, lost their job, company inserted a new firewall, etc.) leaving us with 24 viable participants. Three of these were used in a preliminary pilot study. We sent out 21 surveys, with 4 participants not responding, leaving us with 17 users completing the study.

4.6.3. Results.

4.6.3.1 Novelty and usefulness. Recall our main design consideration for the recommender system was for it to produce useful and novel recommendations. We used responses to Q1 to measure novelty, and responses to Q2 to measure usefulness. Repeated measure analysis of variance showed a significant difference in average usefulness for technique ($F_{2,32} = 13.340, p < .0001$). The ratings were 2.82 for *Linton*, 3.18 for *User-Based*, and 3.44 for *Item-Based*. Pairwise comparison using Bonferroni adjustment showed there was a significant difference between *Linton* and *Item-Based* ($p = .0001$) and *User-Based* and *Item-Based* ($p = .038$). The effect of technique on novelty ratings did not reach significance. The ratings were 3.04 for *Linton*, 2.97 for *User-Based*, and 3.04 for *Item-Based*.

As discussed in the design considerations section we are interested in the quality of the individual recommendations (Figure 2), particularly those falling into the "good" or "poor" categories. As such we do not only want to look at usefulness ratings, but rather judge the quality of the lists which the algorithms provide by assessing the number of good and poor recommendations which each produces.

4.6.3.2 Good and poor recommendations. First, we consider good recommendations to be those where the user was not previously familiar with the command, but after seeing the suggestion, will use it. This corresponds to a response of *strongly disagree*, *somewhat disagree*, or *neither agree nor disagree* to Q1, and a response of *somewhat agree*, or *strongly agree* to Q2. We define the percentage of good recommendations as the average number of recommendations produced which were good.

Repeated measure analysis of variance showed a main effect for the algorithm ($F_{2,32} = 12.301, p < .0001$) on percentage of good recommendations. The overall percentages of good recommendations were 14.7% for *Linton*, 27.2% for *User-Based*, and 30.9% for *Item-Based*. Pairwise comparison using Bonferroni adjustment showed that both *User-Based* ($p = .006$) and *Item-Based* ($p = .001$) had a significantly higher percentage of "good" suggestions than Linton's approach, but the difference between *Item-Based* and *User-Based* was not significant (Figure 10).

We defined poor recommendations as those where regardless of previous familiarity, the user would not use the command, corresponding to a response of *strongly disagree*, or *somewhat disagree* to Q2.

Repeated measure analysis of variance showed a main effect for the algorithm ($F_{2,32} = 11.486, p < .0001$). The overall percentages of poor recommendations were 41.9% for *Linton*, 32.4% for *User-Based*, and 22.1% for *Item-Based*. Pair wise

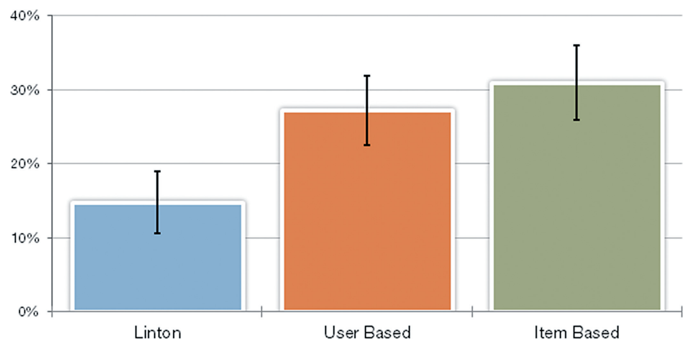

Fig. 10. Percentage of “good” suggestions by technique. Error bars show standard error.

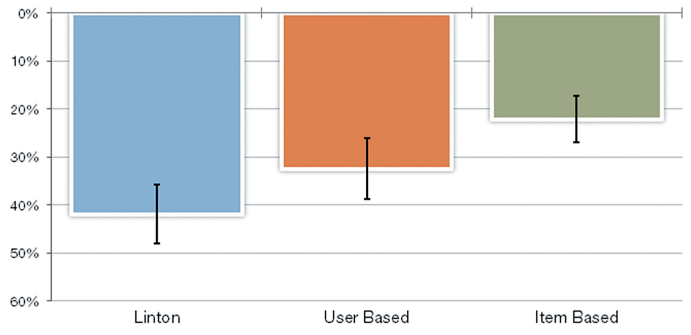

Fig. 11. Percentage of “poor” suggestions. Error bars show standard error.

comparison showed that *Item-Based* was significantly lower than both *User-Based* and *Linton* ($p < .05$), but *User-Based* was not significantly different from *Linton* (Figure 11).

Overall, these results are very encouraging. Compared to Linton’s algorithm, the item-based algorithm increased the number of good commands a user would receive by 110.2%, while reducing the number of poor commands by 47.3%, and in both cases the difference was significant. The user-based algorithm also showed promise, increasing good commands by 85.7%, and decreasing poor commands by 22.6%, although these differences were not significant.

We analyzed how similar the recommendations produced by different algorithms were. Each algorithm generated 8 commands for 17 users, or 136 commands in total. There were 31.6% out of 136 commands recommended by all three algorithms. Item-based and user-based recommended 75 of the same commands (55.1% overlap), Linton and user-based recommended 70 of the same commands (51.4% overlap), and item-based and Linton recommended 38 of the same commands (27.9% overlap). As a special case of user-based approach, Linton’s method defines the neighbor of the active user as all users instead of similar users. It explains why Linton’s and user-based recommendations had the highest degree of overlap.

Overall the item-based approach provided a stronger set of recommendations in comparison to the user-based. We believe this is partially due to the fact that user-based approach accumulates rating for each command from all neighbors. It is thus more likely that commands with high popularity and frequency are recommended. Alternatively, item-based approach builds item-by-item correlation only based on those users who have used the same command pairs, in that way item-based recommendations are more correlated with the active user’s command usage. For example, consider the case of two commands: a very popular command A and a less popular command B; both A and B has not been used by the active user, but B is often used together with an active user's command C. For the user-based approach, A may be recommended because most of the active user's neighbors may have used A. For the item-based approach, B is more likely to be recommended if the active user uses C frequently, because B has more correlation with C. Although the *cf-iuf* rating partially solved this issue for user-based approach, both offline and online study results confirmed that the user-based algorithm still recommends more high frequency commands than the item-based algorithm.

4.7. Subjective Feedback

After we finished our study, we went on a workplace site visit to interview two of the participants who had completed the study. The first participant is a manager who trains new employees and serves as a gatekeeper for learning new features. The second participant reports to the manager. We asked both participants to comment on every item in their personalized recommendation list of commands from the survey. In a few cases we found that a recommended command was known to the user but not used as it had a known limitation for their needs and an alternative workflow was used. We also found two cases where an upgrade rule could have been defined to prevent a poor recommendation. Overall, these two participants felt that receiving such recommendations could be a useful way to learn new features in the software. In particular, the manager thought this could be a very effective way help new employees learn features that they are expected to use in that particular office.

As further anecdotal evidence for recommendations to be potentially useful for learning, we received an unsolicited email from one of our participants three months after the study, which read:

> I just wanted to thank you because I have found myself using three of the recommended commands fairly often these last few months and they've been quite helpful. I'm really looking forward to seeing this feature in the future assuming you are still working on it.

4.8. Summary

In this section we described our implementation of two new collaborative filtering algorithms used to provide software command recommendation algorithms. We also discussed the difference between item-based and user-based approaches. The results of our offline and online evaluations indicate that using such algorithms can significantly improve the nature of recommendation that a user receives, with item-based recommendations performing best.

Based on this promising result, we are motivated to further our investigation into the design space of such recommendation algorithms. In particular, the algorithms described in this section provide long-term or global recommendations, based on a user's entire command usage history. Still open for exploration are algorithms that can provide users with short-term recommendations, based on the user's current context of usage. We investigate this further in the next section.

5. DESIGN SPACE OF CONTEXTUAL COMMAND RECOMMENDATIONS

The algorithms described in Section 4 produce recommendations tailored to an individual. This was shown to have an added benefit over Linton's OWL system; commands which are not relevant to the individual's workflow will be avoided. For example, among AutoCAD users, there is often a distinction in jobs between 2D CAD drafters and 3D designers. It is not useful for a recommender to suggest 3D rendering commands to those 2D users. However, because a user's context is taken into account, a user who

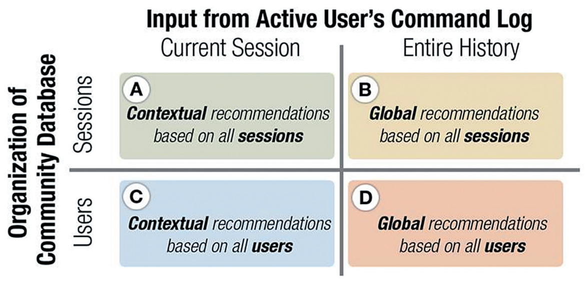
Input from Active User's Command Log
Current Session
Entire History
Organization of Community Database
Sessions
Users
A
Contextual recommendations based on all sessions
B
Global recommendations based on all sessions
C
Contextual recommendations based on all users
D
Global recommendations based on all users

Fig. 12. Design space of command recommendation systems.

uses both 2D and 3D tools may receive commands related to 3D tools even if they are currently drafting in 2D. In this section, we explore the design space for generating dynamic recommendations, and describe our development and evaluation of both contextual and transaction-based recommendation algorithms.

We explore two important dimensions related to contextual recommendations: the scope of the active user's command history for defining current context (current session or entire history), and the granularity of the global command history for generating command-by-command correlations (session-based, or user-based) resulting in the design space shown in Figure 12.

5.1. Input from Active User's Command Log

In the first axis of the design space (horizontal axis in Figure 12), the duration of the active user's command history, used as input to the recommender, is considered. In this dimension we define two discrete values: *current session*, and *entire history*.

5.1.1. Current Session (A and C). Current session command data represents the user's session-specific preferences which are transient and dependent on the current task flow. For example, in an e-commerce recommender system, an item which was purchased recently by a user should have a greater impact on the prediction of the user's future behavior than an item which was purchased many years ago. Similarly, a user's preference may change when switching between different tasks or stages in the workflow; or, when working on multiple projects each of which require different sets of functionality from the application. A sequence of commands that have recently been used by a user should have a greater impact on predicting this user's future action or needs than commands from longer ago. In a command recommender, the user's short-term requirements may change in different sessions.

5.1.2. Entire History (B and D). Alternatively, we can look at the user's entire command history, allowing the system to infer some of the user's stable preferences which remain true throughout. Recommendations generated using long-term history are more stable than recommendations based on session data, but this could mean the recommendations which may be useful at some point in the future, may not be very useful given the current user task. For example, when using AutoCAD a user's complete command stream may indicate interest in both the *3D Alignment* and *2D Alignment* commands, even though the user is currently working in a 2D drafting project when clearly the 2D command is more relevant to the current task. Using the entire history also has the potential advantage of having more data on an individual user to use when computing similarities.

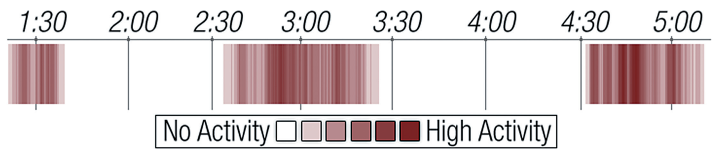

Fig. 13. Example user history of a typical user showing distinct sessions of activity with breaks in activity in between. Red lines indicate the intensity of user activity (darker red means more activity) and the white areas indicate lack of activity.

5.2. Organization of Community Database

The second axis of the design space (vertical axis in Figure 12) determines how we generate command-by-command correlations. We split up the community database by either *sessions* or *users*.

5.2.1. By sessions (A and B). Session-based correlation represents the similarity between two commands which have happened together in a relatively short time frame. Commands are tightly correlated because they have usually happened in the same stage of a user's task flow. For example, command pairs *COPY-PASTE* and *REDO-UNDO* are closely related in the session-based similarity matrix. Of course any command correlations which would be seen over time periods spanning more than one session cannot be captured by using a session-based correlation method.

5.2.2. By users (C and D). In contrast, to capture correlations which can occur over multiple sessions, an alternative is to generate a user-based similarity matrix based on each user's entire command history. User-based correlation generates a matrix containing similarities over the long-term among the items. In Section 4, our global models are generated based on each user's full command history. The benefit of using the user's entire history is to identify similar commands which may be used across different sessions.

5.3. Framework and Algorithms

To the best of our knowledge, there is no previous work exploring the above-described design space of the command recommender domain. We explored four algorithms, each designed to satisfy one of the four quadrants in Figure 12.

5.3.1. Defining a Command Session. One important element is the need to define a command session. When we look at the command inputs of a typical usage scenario we notice that the commands are not distributed evenly over time. For example we can see the activity of a typical user in Figure 13, which shows that even though the application was open the entire time, there were three time periods of relatively heavy activity with two distinct gaps between them which splits this user's command history into three "sessions."

Our command dataset includes the time intervals between every pair of sequential commands performed, resulting in 40 million command pair timings. This data shows that 95% of the command intervals are shorter than one minute and 0.6% of the intervals are longer than one hour. There is also a command set, Q, containing commands such as *QUIT*, *CLOSE*, and *QSAVE*, which end a command sequence immediately. Based on both the command interval and the sequence ending commands, we define a command session:

$$s = \{c_1 t_1 \ldots c_{n-1} t_{n-1}, c_n\}, \forall : t_i < T, c_i \notin Q,$$

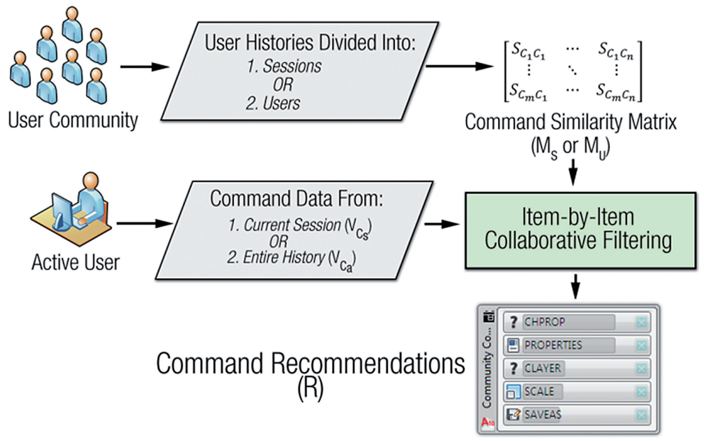

Fig. 14. The recommendation process with the two decision points of using user histories broken up into sessions or users, and looking at the command data from the current session or the entire history of the active user.

where c_i is the i^{th} command in the sequence and t_i is the time interval between c_i and c_{i+1}. T is the amount of time of inactivity between two commands where we consider a new session to begin. We define $T = 1$ *hour* for the purposes of this work.

5.3.2. Algorithms. Based on the results from Section 4, we use the item-by-item collaborative filtering in this section, but the design concepts could be extended to any collaborative filtering algorithm.

Let us assume that we have n users in U, l sessions in S, and a set of m commands, C. Let $u_a \in U$, be referred to as the active user, u_a who will have used ("implicitly rated") a set of commands $C_a \cup C$ in u_a's entire command history, and u_a rates commands in C using a rating function $r_a : C \to [0, I]$, where I is the maximum rating value defined by the system. Let $C_s \cup C_a$ refer to the set of commands used in u_a's last session.

We define $rs : S \times C \to [0.I]$ as the rating function for session-based correlation and $ru : U \times C \to [0, I]$ as the rating function for user-based correlation. We have two similarity matrixes: M_s where $M_s(i, j) = \cos(\theta_{\overrightarrow{rs_{c_i}}, \overrightarrow{rs_{c_j}}})$ and M_U where $M_U(i, j) = \cos(\theta_{\overrightarrow{ru_{c_i}}, \overrightarrow{ru_{c_j}}})$ using session's data and user's data respectively.

A recommendation set R can then be generated for each of the four design quadrants from Figure 12. We use commands in C_s or C_a as an "active list" for the active user; for every unused command, we compute the mean of its similarity to all the commands in the active list; then we generate the top N commands with the highest means to produce the *top-N* recommendation list.

Figure 14 summarizes the recommendation process and the design space. Specifically, it shows the two decision points of how to treat the history files from the user community (to create a command similarity matrix), and which command data to look at for the active user (as input to the item-by-item collaborative filtering algorithm) to generate command recommendations. These two decisions lead to the four quadrants of the design space.

5.3.3. Offline Evaluation. We use the k-tail technique (with k = 1) to evaluate the recommendations generated from algorithms A through D. To evaluate contextual recommendations based on current session data (A and C), we first divide the active user's commands into training and testing sets; we then further divide the training set into two parts so that the second part only includes commands in the last session of the training set. We consider the second part of the set as contextual information for the

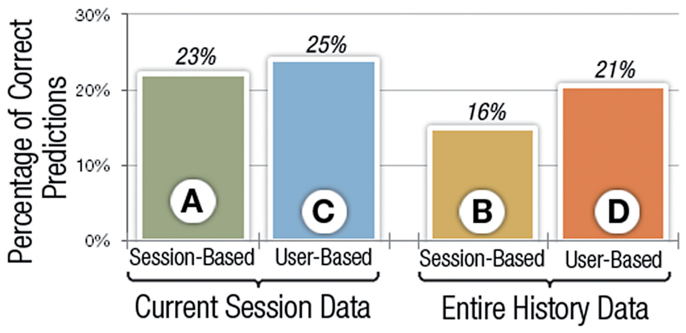

Fig. 15. Offline evaluation of 4 recommendation approaches described in Figure 12 (higher percentages are better).

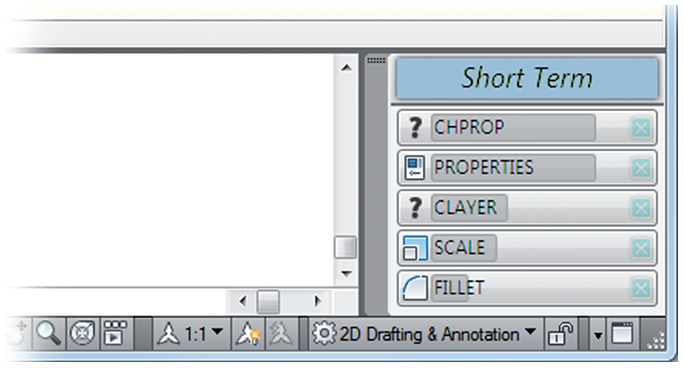

Fig. 16. CommunityCommands palette docked in the bottom right corner beside the command line.

active user. Used commands are filtered out based on the active user's entire command history. Figure 15 shows that a user's more recent commands (A, C) seem to hold more relevance than commands that a user used any time in their history (B, D). This result could be explained by the fact that the user's last session's actions are more likely to be related with the user's action in the near future than those actions happened long time ago.

In the dimension of organization of the community command database, algorithms using user-based data (C and D), showed improvements in comparison to the algorithms using session data (A and B). We believe this is because command pairs used by the same user in separate sessions are not captured when only using session data.

This offline evaluation indicates the short-term contextual recommendations will improve the quality of the commands recommended to the users. However, an online evaluation is necessary to further validate this finding. To perform such an evaluation, we first need to develop an interface that can provide contextual, real-time recommendations to our target users. In the next section, we describe CommunityCommands, the plug-in for AutoCAD that we developed to deliver the recommendations. We then describe the online evaluation we conducted using this plug-in, to further investigate the differences between short-term and long-term recommendations.

6. DESIGN AND EVALUATION OF IN-APPLICATION RECOMMENDATIONS

We designed a working user interface to implement our recommender algorithms and deliver the recommendations to the user (Figure 16). Using the plug-in, we are able to conduct a field study with 32 real users, in their own working environments and within

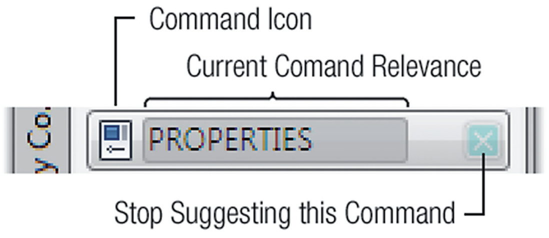

Fig. 17. Components of a suggestion button.

the target application. The study was counterbalanced with 16 users starting in the short-term mode, and 16 starting with long-term.

6.1. User Interface Design Considerations

In the user interface design we adhered to the following guidelines.

1. *Unobtrusive.* The interface should stay out of the user's way. We avoid a system that pops up or forces the user to respond to the recommendation before continuing to work, since this type of system could be frustrating [Hudson et al. 2003; Xiao et al. 2004].
2. *In Context.* The system should provide the recommendations within the application [Knabe 1995]. This way a recommendation can be viewed and tested with minimal switching cost.
3. *Minimal Cost for Poor Suggestions*. The interface should make the task of dealing with poor suggestions, if they do occur, lightweight to minimize frustration, allowing the user to spend more time looking at the good suggestions.
4. *Self Paced.* The user should be able to act on the recommendations when it is convenient for them.

6.2. CommunityCommands Plug-In Design

The CommunityCommands interface was developed as an AutoCAD plug-in using the ObjectArx API. The command recommendation window is contained within a docking palette which can be docked to the side of the application window or left as a floating window over the application. The position and size of the palette can be controlled by the user.

Each of the command buttons contain the command's name and icon, a bar representing the estimated relevance of the command to the user's current context, and a button to dismiss the recommendation to prevent it from appearing in the future (Figure 17). We used the commands iconic representation from AutoCAD's existing menu and toolbar systems, to help users subsequently recognized and locate the commands in the standard AutoCAD UI. A question mark icon was used for commands without iconic representation. Hovering over the command button causes the standard AutoCAD tooltip to appear, and dwelling longer reveals an extended tooltip with additional usage information. Clicking on the command button executes the command.

When commands are added or removed from the list, or the commands already in the list are reordered, a 0.5 second animated transition is used, providing a subtle cue to the user that their recommendation list is changing, and there might be new commands of interest.

In order to generate the recommendations it is necessary to have a command usage history for each user. The plug-in records this data locally to a text database. Besides storing command history information, the plug-in also records when tooltips are activated on the command palette, and when the "close" button is clicked on a command.

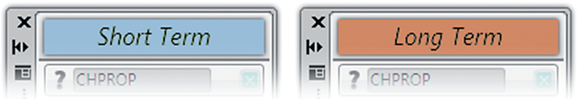

Fig. 18. User interface change between short-term and long-term modes.

The state changes of the recommendations list are recorded so we can see which commands were in the recommendation list at any time, as well as see when the commands in the list were reordered. When the plug-in is installed, the rating matrix, based on the entire user community, is copied to their local machine, so that recommendations can be generated in real-time, without a connection to the internet.

6.3. Online Evaluation

Using this plug-in, we can now evaluate the new short-term algorithms described in Section 5, to see if such algorithms will outperform the long-term algorithms which we evaluated in Section 4. Based on the results from our offline evaluations, we selected the most promising short-term technique, *C: Contextual based on all users,* and the most promising long-term technique, *D: Global based on all users*, and tested them with real users in their real working environments. From here forward technique "C" will be referred to as "short-term" and technique "D" as "long-term."

6.3.1. Participants. We recruited 52 full time users of AutoCAD. To qualify, participants were required to use AutoCAD for more than 20 hours/week. As is often the case with longitudinal offsite studies, we lost some participants during the study due to job changes, lay-offs, etc. and ended up with 32 users completing the 6 week study.

6.3.2. Procedure. For the first 4 weeks of the study a plug-in was installed to record the commands being used by each participant. This plug-in contained no user interface and ran in the background logging the commands used to a local file on the user's computer.

For the next 2 weeks of the study, the users installed a second plug-in which contained the user interface component and the matrices and algorithms necessary to generate the command recommendations. Users were given instructions on how to use the palette, and advised to use it as if it were a normal part of their application. That is, they could interact with the palette as they saw fit and they were not forced to use it. For one week users were shown the short-term recommendations and for the other week they were shown the long-term recommendations.

6.3.3. Design. The palette was designed such that it could not be resized below a size of 160 × 140 pixels and at all times would show a list of exactly 5 recommendations. Users were asked to place the palette in a position within their normal AutoCAD workspace and in a location that was convenient for them. After the first week of using the palette, the recommender changed modes accompanied by a corresponding change in the appearance of the top of the interface which was the only visible difference between two modes (Figure 18).

The study was counterbalanced with 16 users starting in the short term mode, and 16 starting with long term. Participants were asked to keep the palette visible for the duration of the study. If the window was ever closed, it would reappear immediately.

6.3.4. Results

6.3.4.1 How often recommendations change. Command recommendations are computed based on the active user's "active list" (see Section 4.3.2). Compared to the long-term mode, the short-term mode uses a more dynamic active list, and therefore

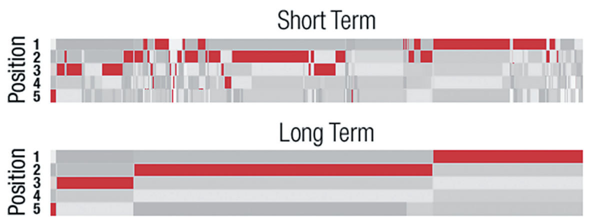

Fig. 19. Visualization comparing short-term and-long term recommendations over a 48 hour time period. In the short-term mode the list of commands is changing often to match the user's context, while in the long-term mode the list is more stable. Here we track the movement of the "COPY" command shown in red.

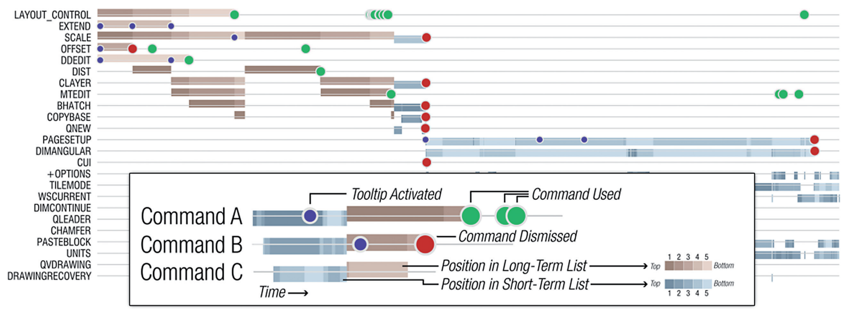

Fig. 20. Overview visualization showing the command recommendations and interactions of a single user over a two week period.

the ordering of commands' similarity score changes more frequently. To get an idea of how those two recommendation modes work differently from the user's perspective we created a visualization to show the state of each type of recommendation list over time (Figure 19). This figure shows the state of the recommendation list over a 48-hour period from an example user. Each one of the 5 rows represents one of the 5 command recommendations that are presented to the user. The COPY command is show in red, and all other commands are shown in shades of grey.

We can see that in the short-term mode there is significantly more reordering of commands, and new commands being introduced while in the long-term mode commands stay positioned in the same spot in the list for a longer period of time. If we consider a list "event" to be any time that a command is added, removed, or changes position in the recommendation list, the short-term recommender averages approximately 120 events/hour where the long-term only generates 13 events/hour.

6.3.4.2 Usage patterns. To get a sense of how individual users were making use of the recommendations, we created an overview visualization for each user (Figure 20). These diagrams show the entire 2 week period of the study in one graphic. All of the commands that ever appeared on the recommendation list are shown in the left hand column. When a command appeared in the recommendation list, the bar to the right of the command name is colored in with a brown bar to show appearing in the long-term list and a blue bar showing the command appearing in the short-term list.

Interaction events with the command window are shown as colored circles: purple circles indicate the tooltip for the command was activated, green circles mark times when the command was used, and red circles show when the user clicked the "×" button to dismiss the recommendation.

By looking at the lines for individual commands among different users, we are able to see some common usage patterns. For example in Figure 21 we see that the user

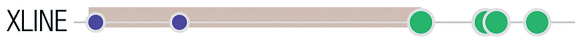

Fig. 21. Command graph showing a user looking at the "XLINE" tooltip several times and then using, and continuing to use the tool.

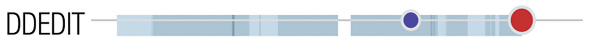

Fig. 22. Command graph showing a user dismissing the "DDEDIT" after fist inspecting the tooltip.

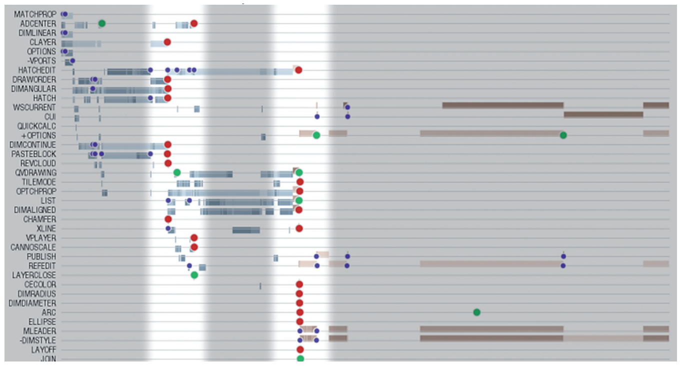

Fig. 23. User history graph showing two "bursts" of interaction with the recommendation window.

had the "XLINE" command in the recommendation list for several days, and looked at the tooltip a couple of times. Eventually the user tries out the command once, which causes it to be removed from the recommendation list. But even after the command is no longer in the list, the user continues to make use of the XLINE command.

In this instance we would say that the user has "adopted" the XLINE command. This is a command they had not used in at least the previous month, and after we suggested it to them, they not only tried it out once, but they continued to use it afterwards.

We also often see situations where a user looks at the tooltip for a command and then dismisses it as in Figure 22. In these cases the user may have been previously aware of the command, or, after looking at the tooltip, determined that it is not useful.

Another pattern we see with many users is "bursts" of activity with the palette such as can be seen in Figure 23. During the two highlighted times the user closes or uses many of the recommendations freeing slots in the list to allow new recommendations to appear. A benefit of the ambient design of the interface is to allow users to interact with the widget when it is convenient for them to do so. We interpret these bursts of activity as evidence that during these two time periods this user had some time to examine what the latest recommendations were and see if they would be useful. If we had forced the suggestions on this user at a different time he may have been too busy to investigate them and just dismissed them immediately.

6.3.4.3 Commands removed. When we look at how the 32 subjects interacted with the interface, we see a wide range of individual differences. For example if we look at how many recommendations each of the users dismissed by clicking on the "×" button associated with a command, we can see that half of the users used the function rarely or not at all, and others used it over 90 times (Figure 24). It appears that some users are interested in manually helping to manage the recommendation list, and others preferred not to intervene.

With this result in mind we think that for this system to be deployed to a large user base it is important to have some type of recommendation "decay" function where

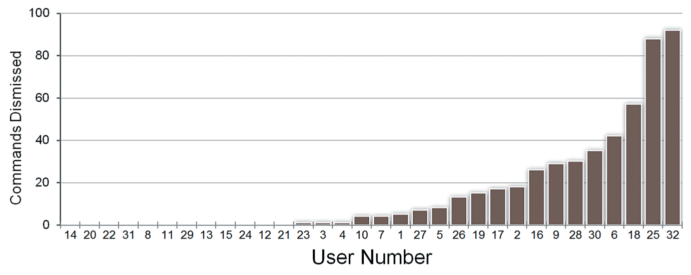

Fig. 24. Number of commands dismissed by each user.

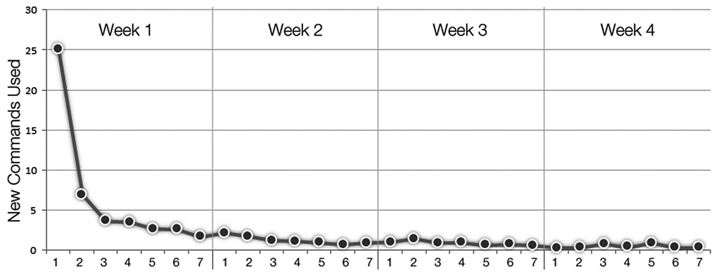

Fig. 25. Average number of previously unobserved commands used each day.

recommendations which have been in the list for an extended period of time are either temporarily or permanently removed from the list. This will allow even users who are averse to interacting with the recommendation list to see additional recommendations.

6.3.4.4 New command exploration. Another metric of interest for the command recommendations is to see how the usage of the palette impacted a user's exploration and usage of commands that they had not previously used. An increase in the usage of such commands would indicate that the plug-in was recommending commands that were useful, or at least of interest to the user. Because we only had 1 month of a user's history, we could not be 100% confident that a previously unobserved command was actually new to the user. However, as illustrated in Figure 25, we can see that the number of new commands issued on each successive day trends downwards. By definition, every command on day 1 is "new," with an average of 25 across all users, and by the 4th week, the users were issuing an average of about one new command per day. This is the average over all users. An individual's curve has more noise, as after some time interval their task may change resulting in a sudden increase in new commands.

By comparing this data, to the data in the 5th and 6th weeks, we can get a sense of the impact that command recommendations had on a user's adoption and exploration of previously unused commands. We first compared the number of new commands used in the 2 weeks before the recommender (weeks 3 and 4) with the number of new commands used in the two weeks with the recommender (weeks 5 and 6). Figure 26 illustrates this analysis by week.

Repeated measure analysis of variance showed a main effect for the week ($F_{3,93} = 7.099, p < .0001$) on number of new commands used. The average number of new commands used was 9.06 and 7.81 for weeks 3 and 4 respectively, and 17.47 and 12.06 for weeks 5 and 6. Pairwise comparison using Bonferroni adjustment showed that *week 5* had significantly more new commands used that either *week 3* ($p = 0.009$) or *week 4* ($p = 0.002$). The other week-to-week comparisons did not reach significance. We repeated the analysis for new commands used at least twice (as opposed to once),

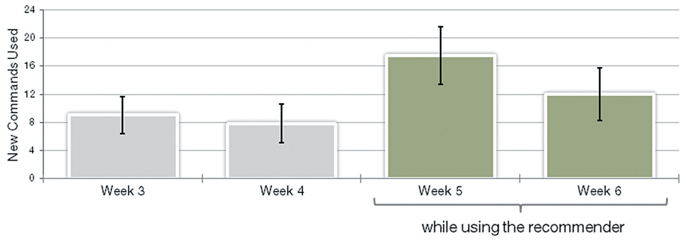

Fig. 26. Average number of new commands used in the last two weeks without, and the first two weeks with, the recommender system.

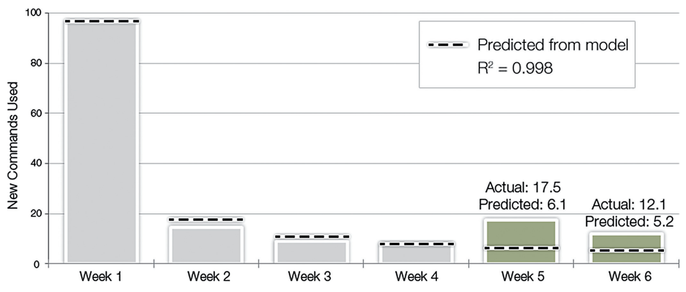

Fig. 27. Number of new commands actually used compared to the model.

and found the same significant differences. That is, there was a main effect for the week ($F_{3,93} = 8.23$, $p < .0001$) and pair wise comparison using Bonferroni adjustment showed that *week 5* had significantly more new commands used twice than both *week 3* ($p = 0.003$) and *week 4* ($p = 0.002$).

Based on the trend illustrated in Figure 25, we can estimate the proportion of the new commands used in weeks 5 and 6 that were due to the introduction of the recommender, and the proportion of new commands that would have been used by chance (without the recommender). The number of "new" commands issued can be predicted by using a binomial or Zipfian distribution (see Appendix for derivation). In Figure 27, the dashed lines show the mean of expected new commands issued in each week using binomial distribution based curve fitting (Equation (3) in the Appendix) across the first four weeks ($R^2 = 0.998$).

This model predicts that 6.1 new commands would be used week 5 and 5.2 commands would be used in week 6 without the recommender (Figure 27). However, with the recommender, the average number of new commands were 17.5 (187% increase) and 12.1 (132% increase) respectively.

Upon further inspection, it was interesting to find that a subset of these new commands had not actually appeared in the recommendation list. In weeks 5 and 6 respectively, an average of 5.6 and 2.4 commands actually appeared in the recommender lists before being used. This leaves 11.9 "extra" commands in week 5, and 9.7 extra commands used in week 6. Because the sum of these recommended and extra commands far exceeds the calculated expected values (Figure 28), we are led to believe that the majority of these extra commands were discovered while investigating or trying out the commands which were recommended. This shows an unintentional benefit of the system: it not only promotes usage of explicitly recommended commands, it can also promote exploration and discovery.

6.3.4.5 Short-term and long-term comparison. To compare the short-term and long-term algorithms we look at the number of recommended commands that were used

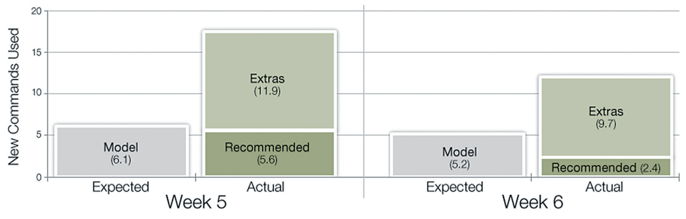

Fig. 28. Average number of new commands used in weeks 5 and 6 split into categories.

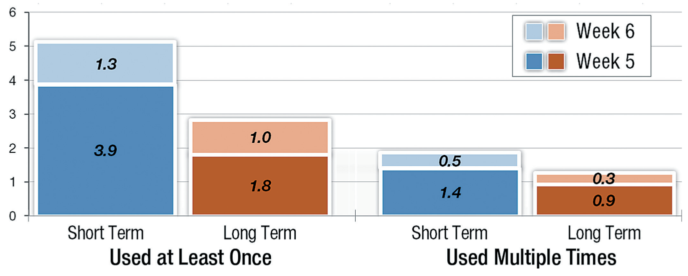

Fig. 29. Average number of recommended commands used once and used multiple times for the short-term and long-term algorithms.

(Figure 29). On average subjects used 5.2 commands suggested by the short-term algorithm, and 2.8 commands suggested by the long-term algorithm. Although the difference did not reach significance (p = 0.113), the trend indicates that short-term recommendations will be more welcomed by users. We repeated the analysis for recommended commands used at least twice and we got the similar result that the short-term recommendation is more preferred by users, but it is not significant. The relatively low number of adopted command per week also shows that people tend to stay with their established software usage skill.

Figure 26 and Figure 28 showed the ordering effects that participants used more new commands in Week 5 than Week 6. In Figure 29, the adopted new commands are broken down by weeks. We further investigated the difference between the two modes through subjective feedback.

6.3.4.6 Subjective feedback. In the postexperiment survey participants were asked which recommendation mode they preferred. 56% preferred the short-term recommendations, while only 16% preferred the long-term (Figure 30). Generally those preferring the short-term mode felt that they were able to see more commands in the list using this mode, and the commands seemed to be more geared towards they work they were currently doing. When asked if they found the interface distracting, only 9% of the users claimed it was distracting due to the list reordering.

Besides introducing brand new commands, several users mentioned that the tool “reminded” them of commands they used to use, but had neglected recently. In this way the tool can be useful even for expert users who may at some point have been familiar with most of the commands in a program, but could use some help in reminding them about commands they have not recently used.

6.4. Summary

In this section, we described a field study, which allowed us to analyze how our in-product command-recommender would be used in real working environments. We had

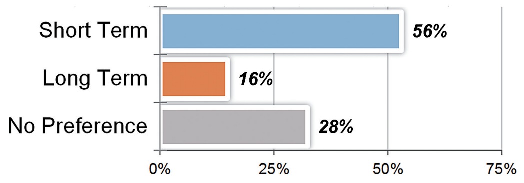

Fig. 30. User preference for recommendation mode.

anticipated many challenges in getting users to adopt any of the recommended commands, since previous research has shown that experienced software users tend to be reluctant to learn new behaviors [Bosser 1987; Linton and Schaefer 2000; Mack 1990]. In particular:

1. Our subjects are all full-time users who use AutoCAD as the main function of their job, many of whom have been doing so for 15 or more years.
2. They tend to be doing work they are familiar with most of the time, and may not need, or feel the need, to learn new tools for that task.
3. Since we performed the study in their real work environments, they are often doing deadline based tasks and may not have time to look at new commands.
4. Even if they do like a command, we do not show them where to access the command in the standard UI.
5. Users were told that they did not need to use our plug-in if they did not want to.

Despite these challenges, our results indicated that the participants did in fact use the tool to explore new commands in the system. In particular, we found a significant increase in the number of new commands used when the plug-in was activated.

Because of the practical difficulties in recruiting professional users, we did not run a control group that never received recommendations, in preference to collecting more usage data from users actually using our tool. Instead, we developed a new model (Appendix) that allowed us to predict the usage rates of such a control group, which showed a high correlation to the observed usage rates from the first four weeks of data.

Finally, a main metric from this study looked at the number of new commands used, and not how many of the recommended commands were actually adopted; that is, how many of these commands will become part of the user's collection of tools they repeatedly use, over an extended period of time. In the scope of a 6 week study, measuring adoption would be difficult. If a recommended command was used many times, we could probably safely say that the command may get adopted, but the question is more complicated for commands which were only used once. A user could try out a command once, find that it is not suitable for their work, and not use it in the future. Alternatively, it may be a type of command that does not typically get used multiple times in a session or even a week, in which case a single use could indicate that the command will be adopted. However, in the absence of an adoption metric, we did repeat our analysis for new commands used at least twice, and found similar trends.

7. DISCUSSION

Our studies have shown that our collaborative filtering command-recommendation algorithms work well compared to previous research approaches, and the plug-in we have developed could be an effective tool to make users aware of relevant commands that they currently do not use. We have validated our work with a series of both offline and online studies which offered promising results. In considering these results, there are a number of topics which should be discussed, with respect to our study findings, and with respect to actual implementation issues.

7.1. Issues Related to Our Studies

In Section 4, we described "good" recommendations as those being both novel and useful to the user. This metric was used as a basis for our evaluations. However, it may also be important to have some of the recommendations on the list be unexpected items that the user would not naturally progress to and may not be assessed as being *useful*, but which expose a new or rare cluster of functionality [McNee et al. 2006b].

With respect to our offline evaluations, we measured the accuracy of different algorithms' ability to predict the next unobserved command based on the training set. However, there is still a subtle difference between new command, which the user has truly never used, and a used command, which has not been observed in the training set. So as with typical recommender system algorithm research, the results of our offline evaluations should be considered with a degree of caution [Herlocker et al. 2004]. However, in both Sections 4 and 6, we were able to validate the results with online evaluations.

Another issue to consider from our results is how well they generalize. Our study participants were professional users, and it would be interesting to understand how our results would generalize to a less-experienced user group, potentially in a learning environment. It is also important to consider how the plug-in would generalize to other software applications and domains. While we do not feel anything would intrinsically change, it might be necessary to reconsider the specifics of the algorithms used, and potentially introduce new types of domain-specific rules.

7.2. Practical Issues Related to Actual Implementations

The work we have presented in this article provides the groundwork for deploying a real tool, and we have in fact developed our research into a working plug-in for AutoCAD that is available for download.[4] However, some practical issues of our research need to be discussed.

One issue to consider is the addition of new commands to the software application; this typically happens with new releases. Software vendors may want to "push" commands into the recommendation list since essentially these commands are experiencing a "cold start" as no users have used the commands yet. Entry points could be determined by product designers or by using the limited beta-customer testing usage patterns that typically precedes a release.

There is also a "cold start" after the user installs the recommender. In this phase, the log of commands is short, and the recommender does not have enough knowledge about the user. One approach for solving this issue is adding a training phase before the recommender starts making recommendations. Ideally, by collecting enough of a user's data over time, the recommender will have enough confidence to tell the difference between the user not knowing a command, and simply not observing the command being used.

Finally, a limitation of the current design is that when a command is used once, it is never recommended again. This could be problematic if a command is accidentally selected. Here we could modify the algorithm to reintroduce the command after looking at how frequently and long ago it was used. Similarly, the user interface could also allow users to choose to dismiss a recommended command temporarily or permanently.

[4]Available from: http://labs.autodesk.com/utilities/acad_community_commands/.

8. FUTURE WORK

There are a number of ways our work could be extended, which we hope to explore in the future. Most of these areas of future work relate to the algorithms which were employed, but we also discuss considerations related to the UI presentation.

A primary area of interest is considering other forms of information to guide recommendations, in addition to a user's command history. For example, we could add additional information related to the user's *context*, such as the type of data in the working file, which type of object is selected, which UI panels are open, or which ribbon tab is active. We could improve the diversity of the recommendations by applying methods discussed by Ziegler et al. [2005]. Also, a potential method of improving accuracy is to apply a weighted combination of multiple algorithms or different preferences.

In addition, the recommendation algorithm could also be modified to be more adaptive. The recommendation algorithm could look at the adoption rate of commands being suggested. This specific information could be fed back into the recommender, where we could overweight the commands that are more commonly adopted by other users. Tracking adoption metrics could also be used to vary the recommendations based on a user's individual skill level, if such information could be distilled. In addition, as often found in other recommender-based systems, we could allow users to provide explicit feedback on the quality of the individual recommendations and feed this into the algorithm.

Alternative recommender algorithms could also be considered. We use collaborative filtering to generate our recommendations; however there are other techniques which could be explored including N-grams and stochastic sequential models. These techniques with additional domain knowledge could be used to more directly model the user's preference and provide more dynamic recommendations. Using the data collected from clicking the "close" button we can extend the system to incorporate critique-based recommendation technique [Faltings et al. 2004]. For example, the plug-in could ask users for their specific reason of dismissing that command, and their feedback will be used to generate new constrains for improving the quality of recommendation.

Another issue which warrants future research is the creation of domain specific rules. We created 21 recommendation filtering rules with a domain expert, which was a relatively straightforward process, lasting approximately 2 hours. However, it may be interesting to try applying modern data mining techniques, to investigate ways to identify these rules automatically from command usage data.

From a broader perspective, our studies involved large bodies of historic usage data, and also rich traces user activity over shorter periods of time. Both of these sources of data present interesting opportunities to aid long term learning goals. In our work we have explored how such data could be used to promote awareness of unused commands, but other opportunities exist, such as profiling user's expertise levels, personalizing learning resources, adapting system features and components, and identifying key, teachable moments.

As one example, future research could investigate recommending higher-level task flows to the user, instead of individual commands. These task flows would contain a collection of commands and sequences of workflows. Similarly, future improvements could inspect short sequences of commands and recommend a single advanced command that could replace the sequence or even an alternative workflow strategy.

Another line of future work is to investigate solutions to the cold start problem, as described in the Discussion section. In e-commerce situations, when new products or services emerge, the interest of customers and the temporal feature of ratings in collaborative filtering may change. To model this concept drifting effect on the dataset of user's ratings, previous work [Ding and Li 2005; Ding et al. 2006; Koren 2009] has used a time-weighted item-by-item correlation to track concept drifting. It would be interesting to apply this same idea to help introduce the new commands in each release of a software package to the users and allow the newer and presumably better commands to be recommended.

The actual interface we designed also warrants further investigation. The goal of our UI design was to promote the awareness of new commands, while minimizing distraction. However, 9% of users did find the plug-in distracting. One alternative design could be adding a button to ask for an update to the recommendations. Or, the command list could remain static until new contextual information the system acquires about the user reaches a certain significance level. Alternatively, users could manually scroll through recommendations if they wanted to.

Another enhancement to the interface is instead of recommending commands by presenting the command name, we could present images of the effect the command has on the user's application data [Terry and Mynatt 2002]. This may give users a better sense of what the unfamiliar command does.

Finally, we hope to perform a longitudinal study using the CommunityCommands UI and recommendation algorithm. Our hope is that the plug-in we have recently released will allow us to analyze such metrics in the future. This would be useful to measure long-term command adoption patterns. In particular, this would allow us to provide richer quantitative data on how receiving recommendations will impact users overall learning experience with a complex software application, and validate some of the initial findings reported in this article.

9. SUMMARY AND CONCLUSION

We have adapted modern recommender collaborative filtering algorithms together with rule-based domain knowledge to promote command awareness and discovery in complex software applications. To test the algorithms offline we developed the k-tail evaluation methodology and then conducted a comprehensive user survey by generating personalized recommendations for a group of real users. Results showed a 2.1 times improvement in the number of good recommendations over previous research.

We further extended this work by exploring the design space of short-term and contextual recommendations, and described a set of algorithms to generate such recommendations. An ambient user interface was designed to present those recommendations to the user, while satisfying our outlined design considerations.

We then conducted a field study of the CommunityCommands plug-in. We deployed and evaluated both long-term and short-term algorithms. Metrics of usage are visualized and discussed, revealing a number of interesting observations. This included a significant increase in the number of unique commands being used after the recommendations were enabled, in the last 2 weeks of the study. It is particularly important to note that these results were obtained while the users were doing production work in their real working environments.

We believe the overall results of our work, in addition to the subjective feedback received, indicate that the integration of our work into software applications is a promising way to improve user's awareness of relevant commands and functionality.

APPENDIX

Here, we model the expected number of new commands which we would expect to observe each week within a single user's command history.

First, we define L_k as the total number of commands executed in week k and n as the number of unique commands. Let $Pr(C_i, L_k)$ represent the probability of command C_i is used at least once in week k. For each individual user, the expected total number of unique new commands used in the first week, $ECount_{W1}$ is equal to the sum of the probabilities that each of the n commands in the application would occur at least once:

$$\mathrm{ECount}_{\mathrm{W}_1} = \sum_{i=1}^{n} \mathrm{Pr}(\mathrm{C_i}, \mathrm{L_1}).$$

The expected total number of unique new commands used in first two weeks can be calculated as the sum of the probabilities that each command would occur in either the first or second week:

$$\begin{aligned}\mathrm{ECount}_{\mathrm{W}_1+\mathrm{W}_2} &= \sum_{i=1}^{n} 1 - (1 - \mathrm{Pr}(\mathrm{C_i}, \mathrm{L_1}))(1 - \mathrm{Pr}(\mathrm{C_i}, \mathrm{L_2})) \\ &= \sum_{i=1}^{n} (\mathrm{Pr}(\mathrm{C_i}, \mathrm{L_1}) + \mathrm{Pr}(\mathrm{C_i}, \mathrm{L_2}) - \mathrm{Pr}(\mathrm{C_i}, \mathrm{L_1})\mathrm{Pr}(\mathrm{C_i}, \mathrm{L_2})).\end{aligned}$$

The expected total number of new commands used only in the second week (and not in the first) is:

$$\mathrm{ECount}_{\mathrm{W2}} = \mathrm{ECount}_{\mathrm{W}_1+\mathrm{W}_2} - \mathrm{ECount}_{\mathrm{W}_1} = \sum_{i=1}^{n} (\mathrm{Pr}(\mathrm{C_i}, \mathrm{L_2}) - \mathrm{Pr}(\mathrm{C_i}, \mathrm{L_1})\mathrm{Pr}(\mathrm{C_i}, \mathrm{L_2})).$$

Generalizing this to any week, k, we have Equation (1):

$$\begin{aligned}\mathrm{ECount}_{\mathrm{W_k}} &= \sum_{i=1}^{n} \left(1 - \prod_{\mathrm{j}=1}^{\mathrm{k}} (1 - \mathrm{Pr}(\mathrm{C_i}, \mathrm{L_j}))\right) - \sum_{i=1}^{n} \left(1 - \prod_{\mathrm{j}=1}^{\mathrm{k}-1} (1 - \mathrm{Pr}(\mathrm{C_i}, \mathrm{L_j}))\right) \\ &= \sum_{i=1}^{n} \left(\mathrm{Pr}(\mathrm{C_i}, \mathrm{L_k}) \prod_{\mathrm{j}=1}^{\mathrm{k}-1} (1 - \mathrm{Pr}(\mathrm{C_i}, \mathrm{L_j}))\right) \qquad (1)\end{aligned}$$

To solve for $\mathrm{ECount}_{\mathrm{Wk}}$, we need to calculate $\mathrm{Pr}(\mathrm{C_i}, \mathrm{L_k})$. We calculate this using the binomial probability formula, based on $\mathrm{p_i}$, command $\mathrm{C_i}$'s overall frequency across the entire user community[5]:

$$\begin{aligned}\mathrm{Pr}(\mathrm{C_i}, \mathrm{L_k}) &= B(L_k, p_i) \\ &= \sum_{j=0}^{L_k} \binom{L_k}{j} p_i^j (1 - p_i)^{L_k - j} \qquad (2)\end{aligned}$$

Based on (1) and (2), we have equation (3)

$$\mathrm{ECount}_{\mathrm{W_k}} = \alpha \sum_{i=1}^{n} \left(\mathrm{B}(\mathrm{L_k}, \mathrm{p_i}) \prod_{\mathrm{j}=1}^{\mathrm{k}-1} (1 - \mathrm{B}(\mathrm{L_j}, \mathrm{p_i}))\right) \qquad (3)$$

This formula assumes that each command occurrence can be any of the n commands in the application. However, users will only know a subset of these commands. Thus, we introduced a single parameter, α, which is an empirically determined constant. We apply this model to the data in Figure 27 by setting α such that $\mathrm{ECount}_{\mathrm{T}_1}$ is equal to the observed number of new commands used in week 1, and setting $\mathrm{L_k}$ to the number of commands used across all participants for each week k.

[5]In our work we know the values of $\mathrm{p_i}$ from our CIP database of command logs. If these values were unknown they could be estimated using a Zipfian distribution [Zipf 1949].

REFERENCES

Adomavicius, G. and Tuzhilin, A. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Engin. 17*, 734–749.

Applecomputers 2009. iTunes.

Baecker, R., Booth, K., Jovicic, S., Mcgrenere, J., and Moore, G. 2000. Reducing the gap between what users know and what they need to know. In *Proceedings of the Conference on Universal Usability.* ACM Press, 17–23.

Bailey, B. P. and Konstan, J. A. 2006. On the need for attention-aware systems: measuring effects of interruption on task performance, error rate, and affective state. *Comput. Hum. Behav. 22*, 685–708.

Basu, C., Hirsh, H., and Cohen, W. 1998. Recommendation as classification: using social and content-based information in recommendation. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98).* AAAI Press, 714–720.

Bosser, T. 1987. Learning in Man-Computer Interaction: A Review of the Literature. Springer, Berlin.

Breese, J. S., Heckerman, D., and Kadie, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98).* Elsevier Science, 43–52.

Canny, J. 2002. Collaborative filtering with privacy via factor analysis. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02).* ACM Press, 238–245.

Carroll, J. M. and Carrithers, C. 1984. Training wheels in a user interface. *Com. ACM 27*, 800–806.

Celma, O. and Herrera, P. 2008. A new approach to evaluating novel recommendations. In *Proceedings of the 2nd ACM Conference on Recommender Systems (RecSys'08).* ACM Press, 179–186.

Chen, L. and Pu, P. 2006. Evaluating critiquing-based recommender agents. In *Proceedings of the 21st National Conference on Artificial intelligence (AAAI'06).* AAAI Press, 157–162.

Claypool, M., Le, P., Wased, M., and Brown, D. 2001. Implicit interest indicators. In *Proceedings of the 6th International Conference on Intelligent User Interfaces (IUI'01).* ACM Press, 33–40.

Cutrell, E., Czerwinski, M., and Horvitz, E. 2001. Notification, disruption and memory: effects of messaging interruptions on memory and performance. In *Proceedings of the 8th Conference on Human-computer Interaction (INTERACT'01).* IOS Press, 263–269.

Cutrell, E. B., Czerwinski, M., and Horvitz, E. 2000. Effects of instant messaging interruptions on computing tasks. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems (CHI'00).* ACM Press, 99–100.

Davison, B. and Hirsh, H. 1998. Predicting sequences of user actions. In *Proceedings of the AAAI/ICML Whorkshop on Predicting the Future: AI Approaches to Time-Series Analysis.* 5–12.

Dent, L., Boticario, J., Mcdermott, J., Mitchell, T., and Zabowski, D. 1992. A personal learning apprentice. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI'92).* AAAI Press, 96–103.

Ding, Y. and Li, X. 2005. Time weight collaborative filtering. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*. ACM Press, 485–492.

Ding, Y., Li, X., and Orlowska, M. E. 2006. Recency-based collaborative filtering. In *Proceedings of the 17th Australasian Database Conference*. Australian Computer Society, Inc., 99–107.

Faltings, B., Pu, P., Torrens, M., and Viappiani, P. 2004. Designing example-critiquing interaction. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI'04).* ACM Press, 22–29.

Farzan, R. and Brusilovsky, P. 2006. Social navigation support in a course recommendation system. In *Proceedings of the 4th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*. Elsevier, 91–100.

Findlater, L. and Mcgrenere, J. 2010. Beyond performance: Feature awareness in personalized interfaces. *Int. J. Hum.-Comput. Stud. 68*, 121–137.

Fischer, G. 2001. User modeling in human-computer interaction. *User Model. User-Adapt. Interac. 11*, 65–86.

Gajos, K. Z., Weld, D. S., and Wobbrock, J. O. 2010. Automatically generating personalized user interfaces with Supple. *Artif. Intell. 174*, 910–950.

Grossman, T., Fitzmaurice, G., and Attar, R. 2009. A survey of software learnability: Metrics, methodologies and guidelines. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems (CHI'09).* 649–658.

Herlocker, J., Konstan, J. A., and Riedl, J. 2002. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Inform. Retri. 5*, 287–310.

Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. 1999. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99).* ACM Press, 230–237.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. 2004. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst. 22*, 5–53.

Hermens, L. A. and Schlimmer, J. C. 1993. A machine-learning apprentice for the completion of repetitive forms. In *Proceedings of the 9th IEEE Conference on Artificial Intelligence Applications*. IEEE Computer Society Press, Los Alamitos, CA, 164–170.

Hill, W., Stead, L., Rosenstein, M., and Furnas, G. 1995. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'95).* ACM Press/Addison-Wesley Publishing Co., 194–201.

Hofmann, T. 2004. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst. 22*, 89–115.

Horvitz, E. 1999. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'99).* ACM, 159–166.

Horvitz, E., Breese, J., Heckerman, D., Hovel, D., and Rommelse, K. 1998. The Lumière project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98).* Morgan Kaufmann, 256–265.

Hsi, I. and Potts, C. 2000. Studying the evolution and enhancement of software features. In *Proceedings of the International Conference on Software Maintenance (ICSM'00).* IEEE Computer Society, 143–151.

Hsu, M.-H. 2008. A personalized English learning recommender system for ESL students. *Expert Syst. Appl. 34*, 683–688.

Hudson, S., Fogarty, J., Atkeson, C., Avraham, D., Forlizzi, J., Kiesler, S., Lee, J., and Yang, J. 2003. Modeling user behavior: Predicting human interruptibility with sensors: A Wizard of Oz feasibility study. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'03).* 257–264.

Igarashi, T. and Hughes, J. 2001. A suggestive interface for 3D drawing. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'01).* 173–181.

Jones, K. S. 1972. A statistical interpretation of specificity and its application in retrieval. *J. Documenta. 60*, 493–502.

Kaufman, L. and Weed, B. 1998. Too much of a good thing?: Identifying and resolving bloat in the user interface. In *Proceedings of the SIGCHI Conference Summary on Human Factors in Computing Systems (CHI'98).* ACM, 207–208.

Knabe, K. 1995. Apple guide: a case study in user-aided design of online help. In *Proceedings of the Conference Companion on Human Factors in Computing Systems (CHI'95).* ACM, 286–287.

Koren, Y. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 447–456.

Krzywicki, A., Wobcke, W., and Wong, A. 2010. An adaptive calendar assistant using pattern mining for user preference modelling. In *Proceedings of the 14th International Conference on Intelligent User Interfaces (IUI'10).* ACM, 71–80.

Lafreniere, B., Bunt, A., Whissell, J. S., Clarke, C. L. A., and Terry, M. 2010. Characterizing large-scale use of a direct manipulation application in the wild. In *Proceedings of Graphics Interface (GI'10).* Canadian Information Processing Society, 11–18.

Linden, G., Smith, B., and York, J. 2003. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Intern. Comput. 7*, 76–80.

Linton, F. and Schaefer, H.-P. 2000. Recommender systems for learning: building user and expert models through long-term observation of application use. *User Model. User-Adapt. Interact. 10*, 181–208.

Liu, J., Wong, C. K., and Hui, K. K. 2003. An adaptive user interface based on personalized learning. *Intell. Syst. 18*, 52–57.

Mack, R. 1990. Understanding and learning text-editing skills: observations on the role of new user expectations. In *Cognition, Computing, and Cooperation*, Ablex Publishing Corp., 304–337.

Matejka, J., Li, W., Grossman, T., and Fitzmaurice, G. 2009. CommunityCommands: command recommendations for software applications. In *Proceedings of the 22nd Symposium on User Interface Software and Technology (UIST'09).*

Mccrickard, D. S., Czerwinski, M., and Bartram, L. 2003. Introduction: design and evaluation of notification user interfaces. *Int. J. Hum.-Comput. Stud. 58*, 509–514.

Mcgrenere, J., Baecker, R. M., and Booth, K. S. 2002. An evaluation of a multiple interface design solution for bloated software. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'02).* 163–170.

MCNEE, S. M., ALBERT, I., COSLEY, D., GOPALKRISHNAN, P., LAM, S. K., RASHID, A. M., KONSTAN, J. A., AND RIEDL, J. 2002. On the recommending of citations for research papers. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'02)*. ACM, 116–125.

MCNEE, S. M., KAPOOR, N., AND KONSTAN, J. A. 2006a. Don't look stupid: Avoiding pitfalls when recommending research papers. In *Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work (CSCW'06)*. ACM, 171–180.

MCNEE, S. M., RIEDL, J., AND KONSTAN, J. A. 2006b. Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *Proceedings of the Conference on Human Factors in Computing Systems.* ACM, 1097–1101.

MILLER, B. N., ALBERT, I., LAM, S. K., KONSTAN, J. A., AND RIEDL, J. 2003. MovieLens unplugged: experiences with an occasionally connected recommender system. In *Proceedings of the 8th International Conference on Intelligent User Interfaces (IUI'03)*. ACM, 263–266.

NGUYEN, Q. N. AND RICCI, F. 2008. Long-term and session-specific user preferences in a mobile recommender system. In *Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI'08)*. ACM, 381–384.

NORMAN, D. A. AND DRAPER, S. W. 1986. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc.

RESNICK, P., IACOVOU, N., SUCHAK, M., BERGSTROM, P., AND RIEDL, J. 1994. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'94)*. ACM, 175–186.

SARWAR, B., KARYPIS, G., KONSTAN, J., AND REIDL, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web.* ACM, 285–295.

SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. 2000. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the 2nd ACM Conference on Electronic Commerce.* ACM, 158–167.

SCHAFER, J. B., KONSTAN, J., AND RIEDI, J. 1999. Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce.* ACM, 158–166.

SHNEIDERMAN, B. 1983. Direct manipulation: A step beyond programming languages. *Comput. 16*, 57–69.

SHNEIDERMAN, B. 2003. Promoting universal usability with multi-layer interface design. In *Proceedings of the Conference on Universal Usability.* ACM, 1–8.

TERRY, M. AND MYNATT, E. D. 2002. Side views: persistent, on-demand previews for open-ended tasks. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST'02).* ACM, 71–80.

TERVEEN, L., MCMACKIN, J., AMENTO, B., AND HILL, W. 2002. Specifying preferences based on user history. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'02)*. ACM, 315–322.

WENG, L.-T., XU, Y., LI, Y., AND NAYAK, R. 2007. Improving recommendation novelty based on topic taxonomy. In *Proceedings of the IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology Workshops.* IEEE Computer Society, 115–118.

WISNESKI, C., ISHII, H., DAHLEY, A., GORBET, M. G., BRAVE, S., ULLMER, B., AND YARIN, P. 1998. Ambient displays: Turning architectural space into an interface between people and digital information. In *Proceedings of the 1st International Workshop on Cooperative Buildings, Integrating Information, Organization, and Architecture.* Springer-Verlag, 22–32.

WITTEN, I. H., CLEARY, J. G., AND GREENBERG, S. 1984. On frequency-based menu-splitting algorithms. *Int. J. Man-Mach. Stud.*, 135–148.

XIAO, J., STASKO, J., AND CATRAMBONE, R. 2004. An empirical study of the effect of agent competence on user performance and perception. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems.* IEEE Computer Society, 178–185.

ZHANG, Y., CALLAN, J., AND MINKA, T. 2002. Novelty and redundancy detection in adaptive filtering. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'02)*. ACM, 81–88.

ZIEGLER, C.-N., MCNEE, S. M., KONSTAN, J. A., AND LAUSEN, G. 2005. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web.* ACM, 22–32.

ZIPF, G. K. 1949. *Human Behavior and The Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley Press, Cambridge, MA.

Received June 2010; revised November 2010; accepted December 2010