Deploying CommunityCommands: A Software Command Recommender System Case Study

Wei Li Justin Matejka Tovi Grossman George Fitzmaurice

Autodesk Research, Toronto, Canada.

firstname.lastname@autodesk.com

Abstract

In 2009 we presented the idea of using collaborative filtering within a complex software application to help users learn new and relevant commands (Matejka *et al.* 2009). This project continued to evolve and we explored the design space of a contextual software command recommender system and completed a four-week user study (Li *et al.* 2011). We then expanded the scope of our project by implementing CommunityCommands, a fully functional and deployable recommender system. CommunityCommands was made available as a publically available plug-in download for Autodesk's flagship software application AutoCAD. During a one-year period, the recommender system was used by more than 1100 AutoCAD users. In this paper, we present our system usage data and payoff. We also provide an in-depth discussion of the challenges and design issues associated with developing and deploying the front end AutoCAD plug-in and its back end system. This includes a detailed description of the issues surrounding cold start and privacy. We also discuss how our practical system architecture was designed to leverage Autodesk's existing Customer Involvement Program (CIP) data to deliver in-product contextual recommendations to end-users. Our work sets important groundwork for the future development of recommender systems within the domain of end-user software learning assistance.

Introduction

Modern computer programs can have thousands of commands available to the user, with a general tendency to increase year after year (Baecher *et al.* 2000). For example, AutoCAD is a widely used software application for both 2D and 3D drafting and design. The number of commands in AutoCAD has been growing linearly and consistently over time. While the growth of commands increases a system's capabilities, the quantity can make learning the system a challenge. In particular, a user's lack of awareness of relevant functionality can act as a barrier to their efficiency with the system (Grossman *et al.* 2009, Shneiderman 1983).

In a "best case scenario", a user would work with an expert next to them who could recommend commands when appropriate (Grossman *et al.* 2009). Indeed, this type of "over the shoulder" learning has been shown to be valuable in the workplace (Twidale 2005), yet it is obviously impractical to assume such assistance would be readily available.

One promising way to address this challenge is to provide users with in-product command recommendations. Existing techniques, such as "tip-of-day" and "did you know", can expose new features, but they may be irrelevant to the user' current task (Fischer 2001, Norman *et al.* 1986). An alternative is to provide personalized command recommendations, based on the user's own history of usage. While some research has been initiated in this area (Linton and Schaefer 2000, Matejka *et al.* 2009), working implementations which deliver these recommendations have never been embedded within a target application. We contribute a recommender system that was released as a plug-in for AutoCAD, and has been used in real usage scenarios. During a one year period of time, over a thousand AutoCAD users downloaded and installed our CommunityCommands plugin from the official Autodesk[1] website as a technology preview.

In this paper, we provide an in-depth discussion of the important issues and challenges we have encountered during the development and deployment of this system. This includes many of the technical details of the recommender system itself, as well as the system architecture and implementation details required to make a real-time command recommender system hosted on a user's local machine, work in practice. In particular, we

[1] http://www.autodesk.com

discuss the key challenges associated with the domain of software functionality recommendations that required us to diverge from the traditional treatment of recommender system problems. This includes: cold start issue; contextual in-product real-time recommendations; and the system architecture to deliver the personal recommendations to end-users, while also protecting user privacy.

In our software command recommender design, we leverage an existing Customer Involvement Program (CIP), which provides a mechanism to collect user command sequence logs anonymously. We also propose a novel architecture that pushes the item-by-item similarity matrix to each user's computer. For users who have privacy and data security concerns, this push model can enable a download-only recommender being deployed to their systems. In addition to privacy concerns, CIP also provides valuable data source for solving the cold start problem. Our hope is that the presentation of these important details will set the groundwork for the future development of recommender systems within the domain of end-user software.

Prior Work

Collaborative filtering based recommender systems have become an important tool to help users deal with information overload and provide personalized suggestions (Hill *et al.* 1995, Shardanand and Maes 1995). Examples include recommending movies (Miller *et al.* 2003), news (Resnick *et al.* 1994), and books (Linden *et al.* 2003). However, little research has been conducted to help users learn and explore a complicated software package using a recommender system. We are aware of two such systems that have been proposed in the literature: OWL for Microsoft Office (Linton and Schaefer 2000) and CommunityCommands for Autodesk AutoCAD (Li *et al.* 2011). The OWL System compares a target user's command frequencies to the average command frequencies of an entire user population. Based on the difference between these frequencies, OWL recommends commands that the target user should use either more or less often. OWL was designed to run within an organization, so it assumes that all users in the community should share the same command usage distribution, and in turn, use the software system in the same way. Across a broad user community, this assumption is unlikely to hold true. Users have different tasks, and preferences, so recommendations should be personalized (Mitchell and Shneiderman 1989). In contrast, CommunityCommands uses personalized collaborative filtering to produce recommendations tailored to an individual (Li *et al.* 2011). This adds a significant benefit over the OWL system; commands that are not relevant to the individual's workflow will be avoided.

In our previous work (Matejka *et al.* 2009, Li *et al.* 2011) we performed several offline evaluations and an online evaluation with a limited number of study participants. In this paper, we describe our deployment of Community-Commands, made available for public download and usage. We provide a detailed description of the system architecture, and report and reflect on the data which was collected from our deployment used by over 1,000 actual AutoCAD users resulting in over 55,000 command recommendations issued over a one year period of time.

Challenges of Building and Deploying a Software Command Recommender System

In this section, we describe a number of challenges that we encountered while preparing our system for public deployment.

Privacy

The issue of user privacy has been explored by recommender system users and researchers (Frankowski *et al.* 2006, Ramakrishnan *et al.* 2001). In many recommender systems, a central server has access to all user profiles and generates personal recommendations. This type of architecture may reveal details about the user, gained through examining their user-item relations. Some privacy research has focused on using a decentralized server architecture combined with strong algorithms to secure user's data (Ahmad and Khokhar 2007, Berkovsky *et al.* 2007, Shokri *et al.* 2009), but this still requires user data to be sent to a network server. The issue of privacy is a significant concern for CommunityCommands. Customers often worry that their usage behaviors and data is being logged. For design software, such as AutoCAD, customer-generated data can be extremely sensitive. In an ideal usage situation, software users should have options and be able to control when to upload their software usage data.

Cold Start

The "Cold-Start" problem is a well-known issue in recommendation systems (Schein *et al.* 2001). For our implementation, we would have no previous data related to the individual user's behavior, and thus, no information to base the recommendations on. It is also difficult to generate the required user-by-user or item-by-item similarity matrices without an existing software usage data set, which results in an inability to draw inferences to recommend items to users. Due to concerns surrounding privacy, it can be difficult to collect the usage data necessary to provide useful recommendations.

In-product Recommendation

Another design challenge of our system is that it provides the recommendations within the product, and they are updated in real-time. This requires the recommendations to be available immediately, unlike previous software recommendation systems in which users receive periodic email updates (Linton and Schaefer 2000). Because of the in-product design and the possibility that the users might not have internet connections, the computations of recommendations must occur locally.

Customer Involvement Program

Many software applications have Customer Experience Improvement Programs[2] (CEIP) or Customer Involvement Programs[3] (CIP) to help collect users' feedback (called *CIP* in the rest of paper). CIP lets users choose to send usage data to the software designers and developers, so they can get anonymous information about how their programs are being used. CIP usually gathers product usage and system configuration information, such as system memory, video card, screen resolution, and operating system details at regular intervals. This type of data is not particularly sensitive. However, in the aggregate, data items such as these give software developers a great deal of insight into what features customers are using, how well they're working, and where they could be improved.

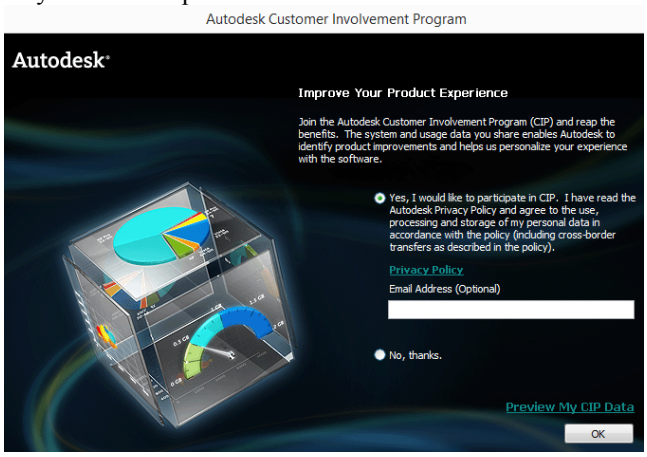

Figure 1. Customer Involvement Program (CIP) Enrollment Interface in AutoCAD 2012.

In AutoCAD, command usage histories are collected as a part of the CIP data. A CIP participation window is presented to AutoCAD users during the software installation process (Figure 1). Sample data and generated reports are also presented to explain that the user's privacy is still being protected. If the user agrees to participate in CIP, when they execute a command, this action is recorded in the form of a (userID, commandID, timestamp) tuple in a centralized database.

The voluntary nature of CIP also provides options to software users to either upload their command log to a central CIP server or keep the log on their computers. AutoCAD users can also turn off CIP anytime by clicking a menu item. Our system leverages CIP to generate command recommendations while users have the option of not revealing their personal information. Before the deployment of CommunityCommands, we used existing CIP data to solve the cold-start problem and implicitly define a rating scheme and generate item-by-item correlations.

Application Description

System architecture

Based on the encouraging results of our one month user study (Li et al. 2011), we developed our recommender prototype into a plug-in for AutoCAD and released it to the public. The system runs as a palette embedded in the AutoCAD workspace, providing within-application and real-time recommendations while a user goes about their normal usage of the software (Figure 2).

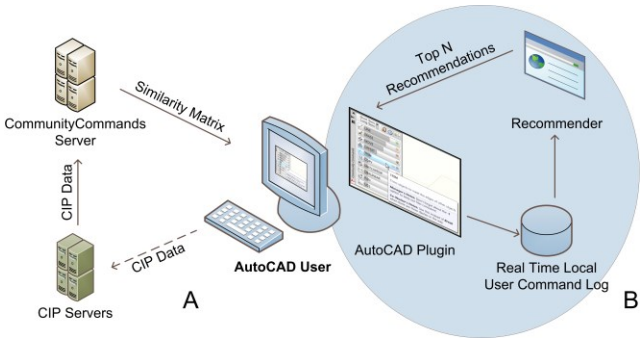

Figure 2. System architecture.

Here we describe our architecture design of this fully functional system for software command recommendations. The system architecture is composed of three components: the user's local machine, the CommunityCommands server, and the main AutoCAD CIP server (Figure 2). When the plug-in is installed and connected to the Internet, a 1.8 MB item-by-item similarity matrix is downloaded (pushed) to the user's local machine, which is used for the item-based recommendation algorithm. The local machine collects the user's command sequence, and computes the recommendations locally each time a new command is issued (using an item-based algorithm). In addition, the CommunityCommands server continuously receives command sequence logs from AutoCAD's main CIP server. This allows us to generate

[2] http://www.microsoft.com/products/ceip/EN-US/default.mspx
[3] http://www.autodesk.com/acip/CIP_Privacy_eng.html

recommendations based on usage profiles of AutoCAD users that may not be running our plug-in. On a monthly basis, the server computes a new item-by-item similarity matrix and each user's local machine downloads and replaces their existing matrix. This system architecture provides two important and unique design properties: preserving privacy and in-product recommendations.

Push based recommendations

For traditional recommender systems, there is no easy way to generate personalized recommendations, without some central system first receiving a user's data. In CommunityCommands, instead of uploading the users' data to the central server, the server pushes the similarity matrix to the user's local computer. Thus, we can still generate a personalized recommendation command list without ever receiving data from that user. The recommendations are still based on the personal data at the local computer, and the aggregated CIP data.

In-product recommendations

Recommended commands are placed in a list within the AutoCAD plug-in palette (Figure 3).

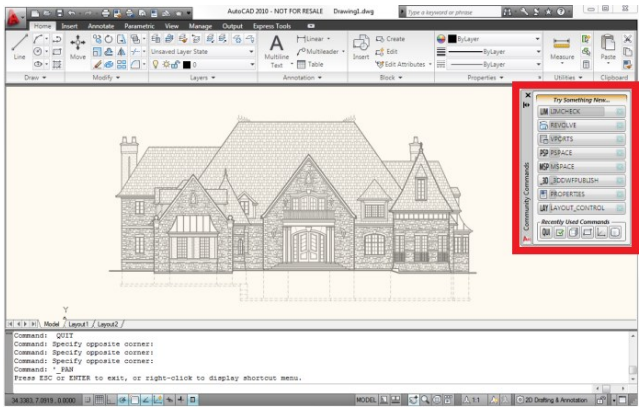

Figure 3. Recommender plug-in palette is opened in AutoCAD.

Clicking on the command button executes the command. If a command in the recommendation list is used, it is immediately removed from the list and displayed in a "most recently used commands" list. Hovering over the command button causes the standard AutoCAD tooltip to appear, and dwelling longer reveals an extended tooltip with additional usage information (Figure 4).

During our development process, we found it critical to be minimally disruptive to the computational resources needed by the main application. Under normal usage, computation of recommendations is unnoticeable to the user, so we compute the recommendations after an individual command has been executed. However, we have to delay the recommender computation if we observe a rapid succession of command usage. In addition, since AutoCAD has a scripting language that can issue multiple commands without user input, we defer processing the recommendations and updating the UI until our threshold idle time of 0.5 seconds has been satisfied.

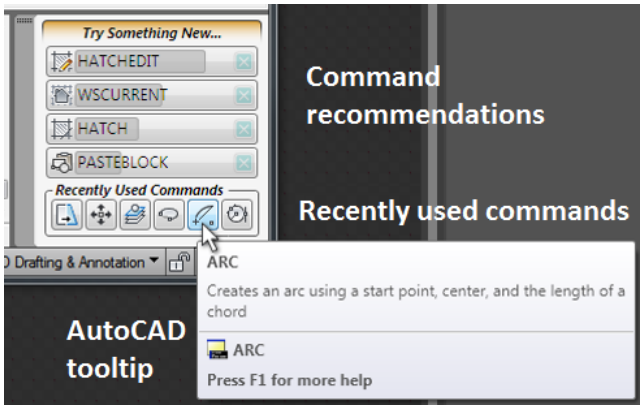

Figure 4. Recommended and recently used commands. Tooltip appears when mouse is hovered over the command.

Training before recommending

To further address the cold start problem, the plug-in begins in a training period, where commands are logged, but no recommendations are presented. Determining the right length of this training period is difficult – we wanted the recommendations to start as soon as possible, but only after we reliably know what commands the user is already aware of. To minimize the time needed for training, we ran a pilot test by analyzing data from 27 users (Li *et al.* 2011). On a daily interval, we measured the rate at which new commands were used (had not been previously observed for that user), across a period of 4 weeks (Figure 5). The data showed that the rate of using "new commands" levels off quickly. For example, after 8 days, 50% of users had less than 3 new commands per day. However, because users will have different daily usage rates, this public released recommender exits the training phase when the user performs less than 3 new commands on two consecutive days. To ensure enough data has indeed been collected, it also requires that the training phase was active for at least 10 usage days, or, until at least 200 commands have been captured.

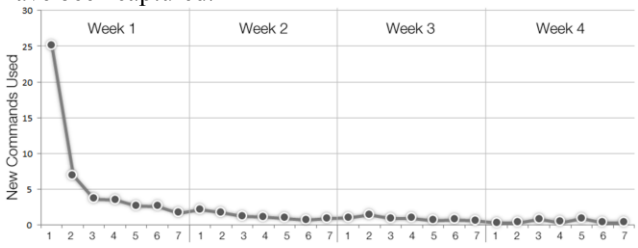

Figure 5. New command adoption rates based on 27 users.

During this training phase, we display a message to the user, and use the pallet to provide access to recently used

commands. This gives the users some value, while waiting for the recommendations to begin (see Figure 6).

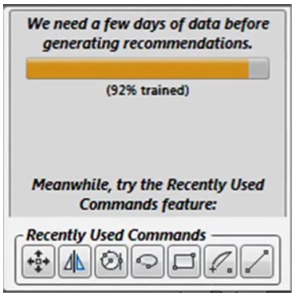

Figure 6. Recommender training phase UI

Use of AI Technology

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

Ratings

Standard collaborative filtering algorithms work by viewing a dataset as a rating matrix. These ratings are either captured implicitly, for example, through purchase records and browsing histories, or explicitly, by asking users to rate the items. We need to map users' command history onto a rating matrix.

One approach is to allow a user to give explicit ratings for each command. This approach would not utilize the user's historical data and would thus suffer from the cold start problem (Schein *et al.* 2001). Moreover, an explicit rating system would be impractical, since software application users will be focused on their primary task, not on rating the functions which they use. In addition, research has shown that users may be reluctant to provide explicit ratings (Shardanand and Maes 1995). As such, the implicit acquisition of user preferences of software commands is more favorable in practice.

Our method uses the command frequency to imply the rating for the user (Li at el 2011). To model how important a command is to a particular user within a community, and to suppress the overriding influence of commands that are being used frequently and by many users, we have adapted *tf-idf* (Jones 1972) into a *command frequency, inverse user frequency (cf–iuf)* rating function. 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 for user u_j, and the denominator is the number of occurrences of all commands for 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 rating in *cf–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.

For each user u_j, we populate 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.

Rather than matching users based on their command usage, our item-based collaborative filtering algorithm matches the active user's commands to similar commands. Similar to user-based approach, each cell, $V_i(j)$, contains the *cf-iuf* value for each user u_j. In our released recommender, we applied item-based approach and customized our suggested commands based on active user's short term preference (session-based command history) to generate contextual in-product real-time recommendations (Li *et al.* 2011).

Novelty evaluation metrics

Command recommendation is a top-N recommendation problem, which identifies a set of N commands that will be of interest to a user (Karypis 2001, Herlocker *et al.* 2004). 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. As such, we were required a metric that would indicate usefulness *and* novelty. To do so, we developed a *k-tail evaluation* which dynamically measures the usefulness of an algorithm based on the sequential information in a user's command log (Matejka *et al.* 2009). Here, we propose to approximate a command recommendation's novelty factor using its binomial probability. We call this the *binomial novelty indicator* (BNI).

To evaluate the novelty of the recommendations, we compute the probability that a command, which was correctly predicted by the recommender, would appear in the testing set by random chance. We do this by using the binomial probability formula, based on a command's overall frequency across the entire user community:

$$P(k) = \binom{l}{k} p^k (1-p)^{l-k}$$

where *P(k)* is the probability of a specific command *C* executed exactly *k* times in a commands sequence of length *l*, and *p* is the overall probability of *C* being executed in the dataset. The cumulative distribution function for *k* can be expressed as:

$$F(k;l,p) = \sum_{i=0}^{k} \binom{l}{i} p^i (1-p)^{l-i}$$

F(l,l,p) represents the chance of seeing command *C* at least once. So we define the *binomial novelty indicator* (BNI) as:

$$B(l,p) = \sum_{i=0}^{l} \binom{l}{i} p^i (1-p)^{l-i}$$

This gives us an explicit measurement as to the likelihood a recommended command would have appeared in the sequence by chance. For example, consider a command *A* that has a frequency of 0.036 and a command *B* that has a frequency of 0.002, across all users, and a testing set with 13000 commands. We compute that there is a 95% chance that *A* appears in the testing set once or more, and a 3% chance that *B* appears once or more. If the recommender predicts both *A* and *B* correctly, we can be reasonably certain that the user more likely knew *A* than *B*. Comparing this across all correctly recommended commands, we can get a measurement of how novel, overall, the commands that a recommender algorithm generates are. Thus, we combine BNI with *k*-tail offline evaluation by computing the mean of BNI for every unique command in $R \cap T$, where *l* is the length of *T*. Our deployed recommender uses both *k-tail* and BNI to select collaborative filtering algorithms and tuning parameters.

Application Use and Payoff

Overall Usage

We report how long the CommunityCommands plugin was deployed on the user's system. This deployment time was calculated using the time stamps of the first and last time the user ran the recommender. During the one year period after we released this recommender system, approximately 1100 AutoCAD users downloaded and installed the plug-in. 983 users used the plug-in for at least one day. 709 users used the plug-in for more than 30 days. On average, the plug-in was installed at the user's computer for more than two months (69.8 days). We also observed that most users who have very short usage times did not pass their training phase before they uninstalled or disabled the plug-in.

Recommendation adoption

Our hope is that users of the recommender system would start using the recommended commands. We hope they not only try the command a few times, but adopt the recommendations into their regular workflows. Figure 7 shows the number of recommended commands being used by the users who have moved past the training phase. The figure contains the recommended commands being used at least once, three times, ten times and twenty times. We call those commands *adopted recommendations* or *useful recommendations*. On average, 21.4 recommendations were used by users at least once. 14 new recommendations were used by users more than three times, 9.6 for 10 times and 7.3 for 20 times.

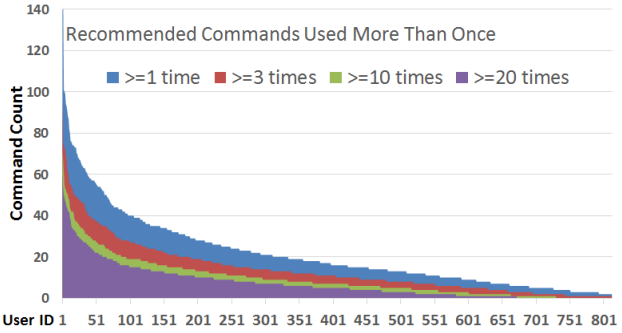

Figure 7. Recommended command adoption

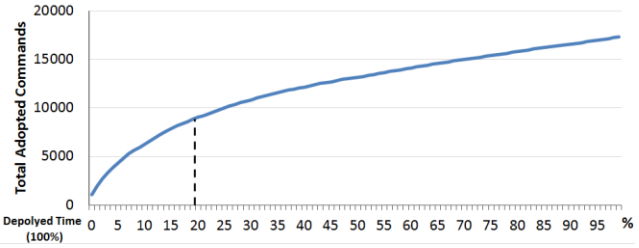

Figure 8. Total adopted useful recommendations over deployed time. The horizontal axis shows the percentage of time passed.

Figure 8 shows the distribution of the total adopted recommendations over time. Here we assume all users start at the same time and spend the same amount time using the system. This figure shows 50% recommendation adoptions happened during the first 19% of the entire period of system usage time.

CommunityCommands only recommend commands that had never been executed in the user's command history. But there may be commands used by the user before the installation of the plug-in. As such, some of these adopted commands may have already been known to the user.

CIP Enrollment

CIP is a key component for solving the users' privacy concerns and cold-start problem. A large group of users (71.3%) who downloaded the CommunityCommands plug-in enrolled in CIP. This of course means that 28.7% of users did not enroll into CIP, mostly due to privacy and technical concerns. As such, our system needs to work for both user groups.

Command usage visualization

To help visualize the data which was collected during our deployment, we developed *Personal Software Usage DNA* diagrams for the users of our plug-in. These diagrams are generated by looking at the command usage patterns of each individual user. By ordering the commands based on the community's overall usage, and coloring them based on the individual's usage, we can see commands that an individual is using more (or less) often than the community as a whole. By looking at how densely the individual row is filled in, we can also see if the individual uses a lot, or relatively few commands.

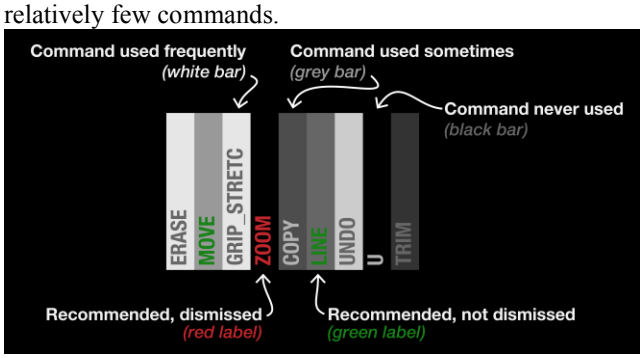

Figure 9. Legend of software usage DNA diagram

Figure 9 presents the information included in each DNA diagram. A red command name means that the command was recommended but was removed by the user from the recommendation list. A green command name means that it was normally showed in the recommendation list. The brightness of the command background represents the usage frequency of that command. So a green command on a bright background is a strongly adopted recommendation. Figure 10 shows the 17 most active users' DNA diagram, with the top user enlarged. In the future, it could be interesting to present these *Personal Software Usage DNA* diagram to the end users, to encourage usage reflection and further command adoption.

Conclusion and Future Work

Based on our experiences, we believe that recommender systems have a rich future for use within software applications. We have provided a detailed treatment of the issues surrounding the development of a command recommender system and the architecture used for its deployment. Our hope is that this research will serve as groundwork and inspiration for future efforts in this area.

We have shown that collaborative filtering algorithms can identify commands that will be useful to a user. This leads us to believe that such systems could also be used to recommend higher-level task flows and relevant tutorial materials.

The item-based collaborative filtering provides relevant and novel recommendations. It aggregates user-item relations into item-item relations. When combined with the system architecture we proposed here, the item-based algorithm can also preserve user's privacy, which is a desirable feature for many business applications.

Certain software applications, including AutoCAD, have a main version, but also "parallel" customized versions for specific user groups. By using collaborative filtering technology, we will be able to recommend customized software features to the appropriate user groups. For example, AutoCAD has vertical versions for mechanical engineers, electric engineers, civil engineers and architects. Recommending commands commonly used by civil engineering to architects, when those commands fit the current workflow, could increase the diversity and novelty of current recommendations.

Another issue is related to software upgrades. In e-commerce situations, when new products or services emerge, the interest of customers and the temporal feature of the ratings in collaborative filtering may change. Previous work (Ding and Li 2005) 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 new commands in each release of a software package to the users and allow the newer and

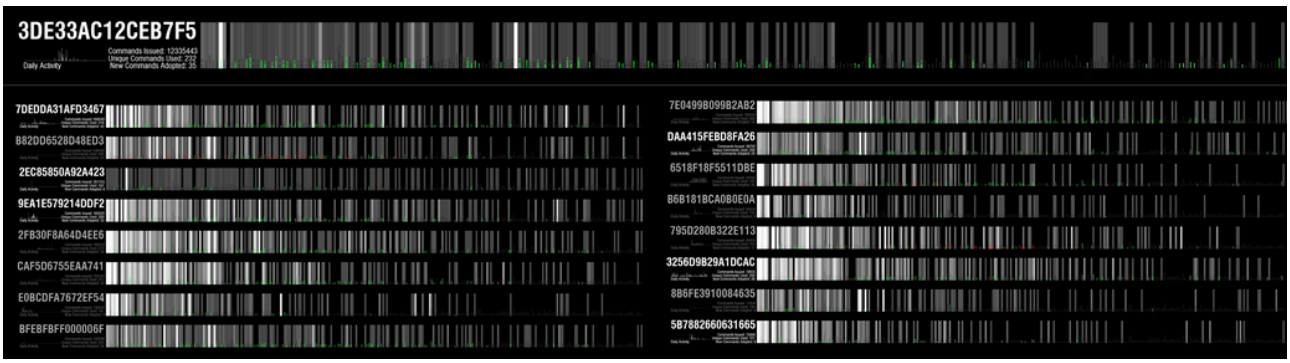

Figure 10. Software usage DNA diagrams from the 17 most active users.

potentially more efficient commands to be recommended.

In summary, the novel contribution of our work is the description of system architecture that has allowed us to embed a software command recommender system within a target application, during real usage situations. Software command/feature recommendation opens a new domain for recommender system research. Many interesting problems arise which open up areas for future work.

Acknowledgement

We would like to thank Joseph A. Konstan for providing valuable suggestions and participating discussions during this project development.

References

Ahmad, W. and Khokhar, A. (2007). An Architecture for Privacy Preserving Collaborative Filtering on Web Portals. *Proceedings of the Third International Symposium on Information Assurance and Security*. 273-278.

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. *Universal Usability-2000*. 17-23.

Berkovsky, S., Eytani, Y., Kuflik, T. and Ricci, F. (2007). Enhancing privacy and preserving accuracy of a distributed collaborative filtering. *Proceedings of the 2007 ACM conference on Recommender systems*. 9-16.

Ding, Y. and Li, X. (2005). Time weight collaborative filtering. *Proceedings of the 14th ACM international conference on Information and knowledge management*. 485-492.

Fischer, G. (2001). User Modeling in Human\–Computer Interaction. *User Modeling and User-Adapted Interaction*. 11(1-2):65-86.

Frankowski, D., Cosley, D., Sen, S., Terveen, L. and Riedl, J. (2006). You are what you say: privacy risks of public mentions. *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. 565-572.

Grossman, T., Fitzmaurice, G. and Attar, R. (2009). A Survey of Software Learnability: Metrics, Methodologies and Guidelines. *ACM CHI conference on Human Factors in Computing Systems.* 10 pages.

Herlocker, J. L., Konstan, J. A., Terveen, L. G. and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst*. 22(1):5-53.

Hill, W., Stead, L., Rosenstein, M. and Furnas, G. (1995). Recommending and evaluating choices in a virtual community of use. *Proceedings of the SIGCHI conference on Human factors in computing systems*. 194-201.

Jones, K. S. (1972). A statistical interpretation of specificity and its application in retrieval. *Journal of Documentation*. 60(5):10.

Karypis, G. (2001). Evaluation of Item-Based Top-N Recommendation Algorithms. *Proceedings of the tenth international conference on Information and knowledge management*. 247-254.

Linden, G., Smith, B. and York, J. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*. 7(1):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 Modeling and User-Adapted Interaction*. 10(2-3):181-208.

Matejka, J., Li, W., Grossman, T. and Fitzmaurice, G. (2009). CommunityCommands: Command Recommendations for Software Applications. *Proceedings of the 22nd Symposium on User Interface Software and Technology*.193-202.

Miller, B. N., Albert, I., Lam, S. K., Konstan, J. A. and Riedl, J. (2003). MovieLens unplugged: experiences with an occasionally connected recommender system. *Proceedings of the 8th international conference on Intelligent user interfaces*. 263-266.

Mitchell, J. and Shneiderman, B. (1989). Dynamic versus static menus: an exploratory comparison. *SIGCHI Bull*. 20(4):33-37.

Norman, D. A. and Draper, S. W., *User Centered System Design; New Perspectives on Human-Computer Interaction*. 1986: L. Erlbaum Associates Inc. 526.

Schein, A., Popescul, A., Ungar, L., Pennock, D. (2001). Generative Models for Cold-Start Recommendations. *the 2001 SIGIR Workshop on Recommender Systems*.

Ramakrishnan, N., Keller, B. J., Mirza, B. J., Grama, A. Y. and Karypis, G. (2001). Privacy Risks in Recommender Systems. *IEEE Internet Computing*. 5(6):54-62.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. and Riedl, J. (1994). GroupLens: an open architecture for collaborative filtering of netnews. *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. 175-186.

Shardanand, U. and Maes, P. (1995). Social information filtering: algorithms for automating "word of mouth". *Proceedings of the SIGCHI conference on Human factors in computing systems*. 210-217.

Shneiderman, B. (1983). Direct Manipulation: A Step Beyond Programming Languages. *Computer*. 16(8):57-69.

Shokri, R., Pedarsani, P., Theodorakopoulos, G. and Hubaux, J.-P. (2009). Preserving privacy in collaborative filtering through distributed aggregation of offline profiles. *Proceedings of the third ACM conference on Recommender systems*. 157-164.

Twidale, M. B. (2005). Over the Shoulder Learning: Supporting Brief Informal Learning. *Comput. Supported Coop. Work*. 14(6):505-547.

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):6:1-6:35