Deploying CommunityCommands: A Software Command Recommender System Case Study

Wei Li, Justin Matejka, Tovi Grossman, George Fitzmaurice

■ *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 six-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 a publically available plug-in for Autodesk's flagship software application AutoCAD. During a one-year period, the recommender system was used by more than 1100 users. In this article, we 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. We also present our system usage data and payoff, and provide an in-depth discussion of the challenges and design issues associated with developing and deploying the software command recommender system. Our work sets important groundwork for the future development of recommender systems within the domain of end user software learning assistance.*

Modern computer programs can have thousands of commands available to the user, with a general tendency to increase year after year (Baecker et al. 2000). For example, AutoCAD is a widely used software application for both two-dimensional and three-dimensional drafting and design. The number of commands in AutoCAD has been growing linearly and consistently over time (figure 1). While the growth of commands increases a system's capabilities, the quantity can make learning the system a challenge. In particular, users' 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, users 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

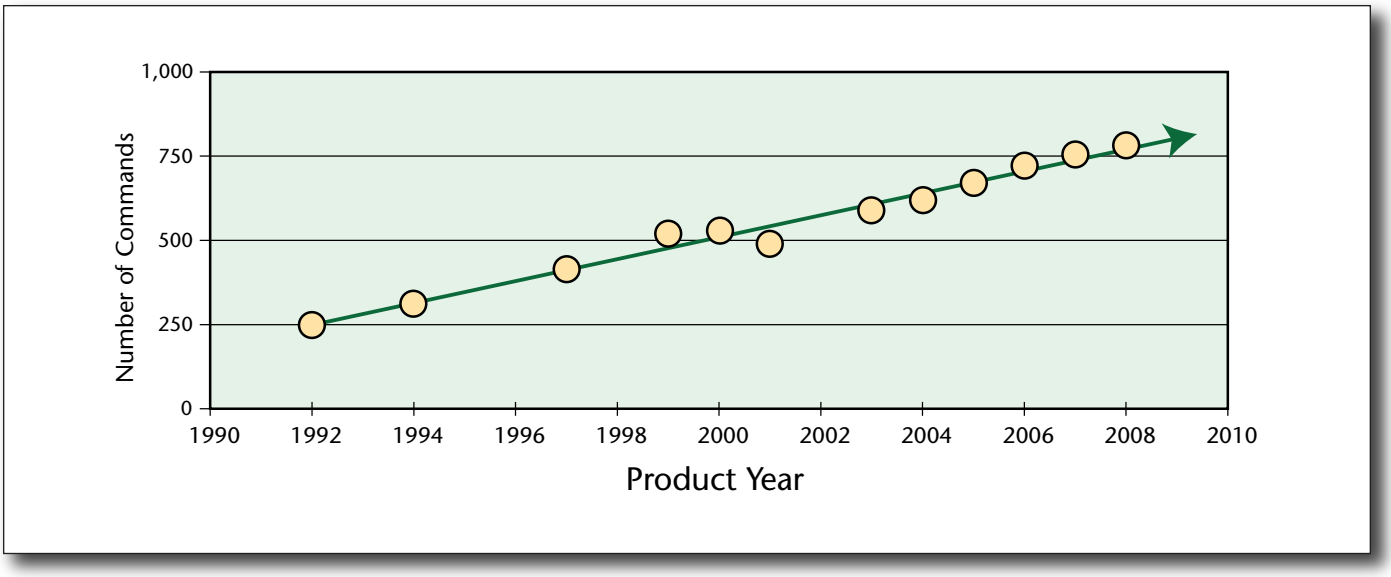

Figure 1. The Number of Commands in AutoCAD.

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's current task (Fischer 2001, Norman and Draper 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 that 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, more than a thousand AutoCAD users downloaded and installed our CommunityCommands plug-in from the official Autodesk[1] website as a technology preview.

In this article, 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 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 issues; 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 a 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, Smith, and York 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, we presented CommunityCommands, a command recommender that uses personalized collaborative filtering to produce recommendations tailored to an individual (Matejka et al. 2009). This adds a significant benefit over the OWL system; commands that are not relevant to the individual's workflow will be avoided. We then continued to explore the design space of a contextual software command recommender system and completed a six-week user study (Li et al. 2011). We found that the recommender successfully exposed users to new commands, as unique commands issued significantly increased.

We then expanded the scope of this research by implementing a fully functional and deployable recommender system, including a publically available plug-in for Autodesk's flagship software application AutoCAD. During a one-year period, the recommender system was used by more than 1100 users. Here we describe our deployment of this system, provide a detailed description of the system architecture, and report and reflect on the data that was collected from our deployment used by more than 1000 actual AutoCAD users resulting in more than 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] to help collect users' feedback (called CIP in the remainder of this article). 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.

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 2). 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

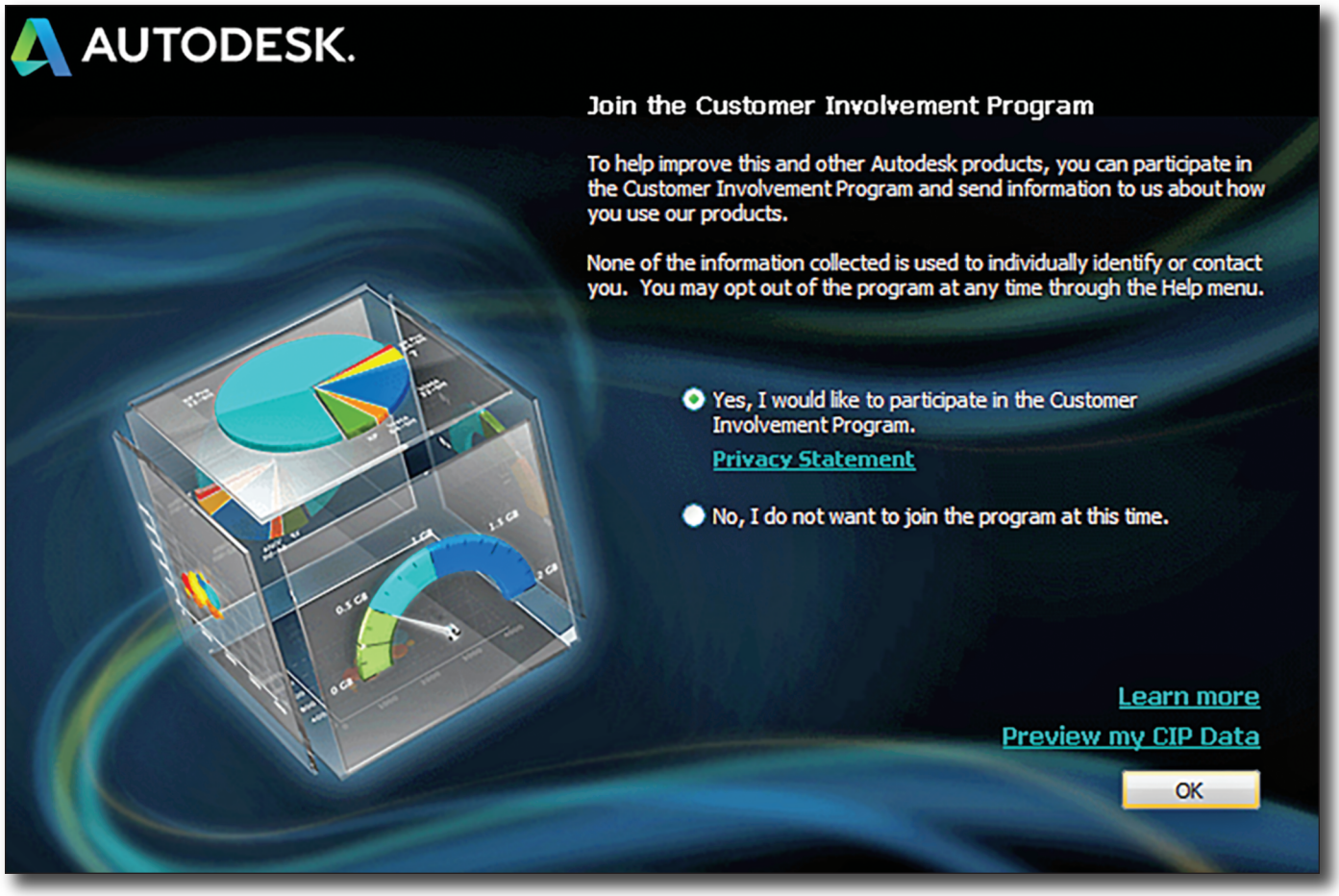

Figure 2. CIP Enrollment Interface in AutoCAD 2012.

(userID, commandID, time stamp) tuple in a centralized database.

The voluntary nature of CIP also provides options to software users either to upload their command log to a central CIP server or to keep the log on their computers. AutoCAD users can also turn off CIP any time 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

Based on our previous work (Matejka et al. 2009, 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 users go about their normal usage of the software (figure 3).

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 3). When the plug-in is installed and connected to the Internet, a 1.8 MB data package is downloaded (pushed) to the user's local machine, which contains a command-to-command similarity matrix used for suggesting commands. The local machine collects the user's command sequence and computes the recommendations locally each time a new command is issued. In addition, the CommunityCommands server continuously receives command sequence logs from AutoCAD's main CIP server. This allows us to update the command-to-command similarity matrix on usage profiles of AutoCAD users that may not be running our plug-in. On a monthly basis, the server computes a new command-by-command similarity matrix and each user's local machine downloads and replaces the existing matrixes. This system architecture provides two important and unique design properties: preserving privacy and in-product recommendations. The following section will discuss how to generate those similarity matrixes.

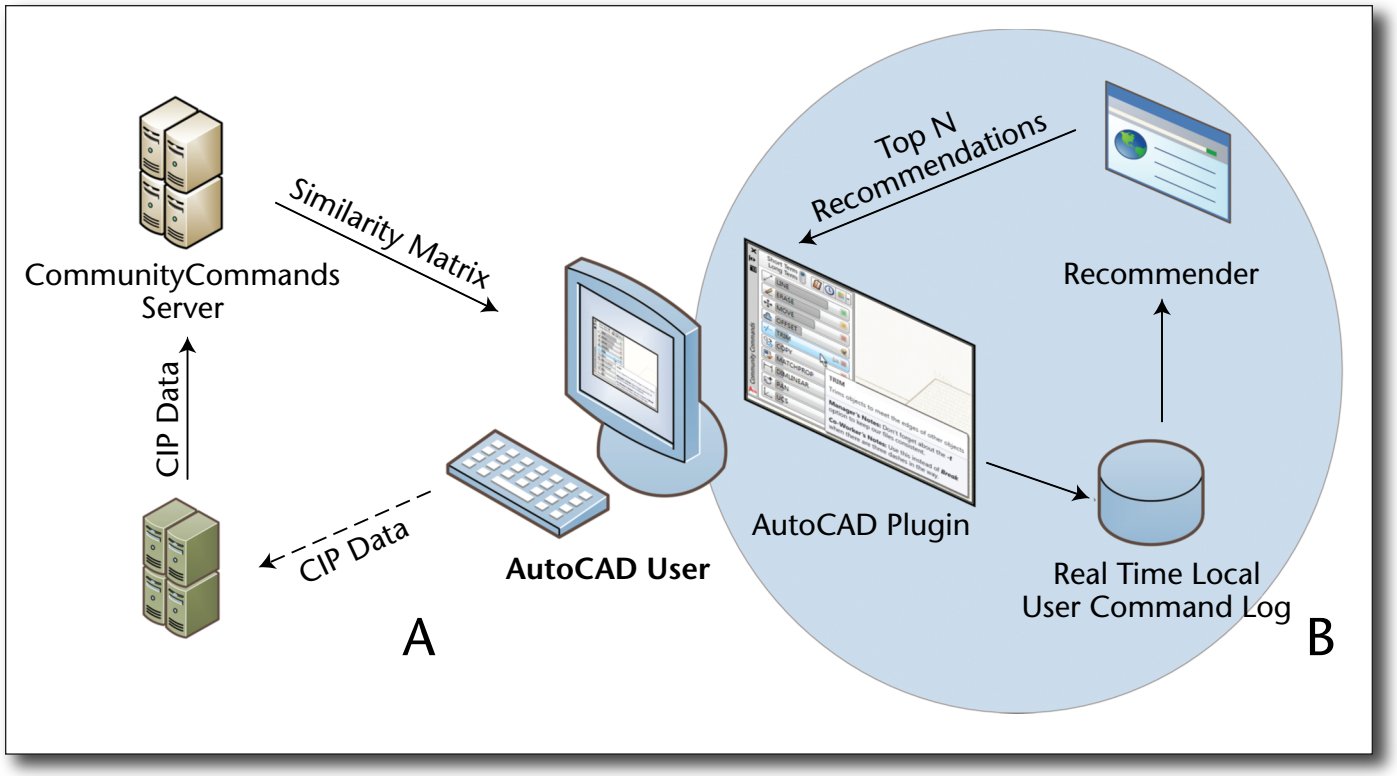

Figure 3. System Architecture.

Use of Collaborative Filtering

We generate a command-to-command similarity matrix using the item-based collaborative filtering approach. 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.

Command Ratings

Standard collaborative filtering algorithms work by viewing a data set 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 that 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 et al. 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:

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

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

$$cf_{ij} \cdot iuf_{ij}$$

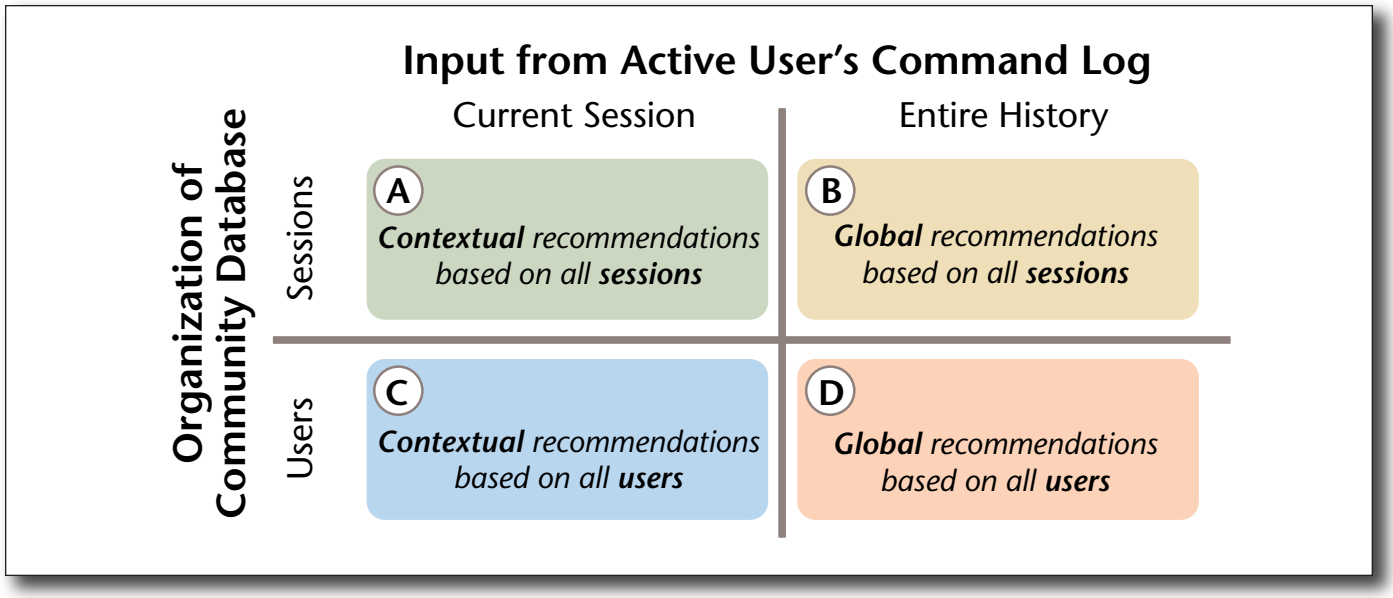

Figure 4. Design Space of Command Recommendation Systems.

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.

Item-Based Collaborative Filtering

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). 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.

Step 1. Defining User Vectors
We first define a vector V_i for each command c_i in the n dimensional user space. Each cell, $V_i(j)$, contains the *cf-iuf* value for each user u_j.

Step 2. Build a Command-to-Command Similarity Matrix
We generate a command-to-command similarity matrix, M. M_{ik} is defined for each pair of commands i and k as:

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

Step 3. 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 in the current session.

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

Step 4. Find Similar Unused Commands
Next, we define a similarity score, s_i, for each command c_i that is not in the active user's active list:

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

Step 5. 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.

The first two steps are performed offline based on data from all users. Then the rest of the steps find the most similar unused commands for each active user based on her/his active list and the command-to-command similarity matrix.

Contextual Command Recommender Design

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 4.

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 5, which shows that even though the application was open the entire time, there were three time periods of relatively heavy activity with two distinct

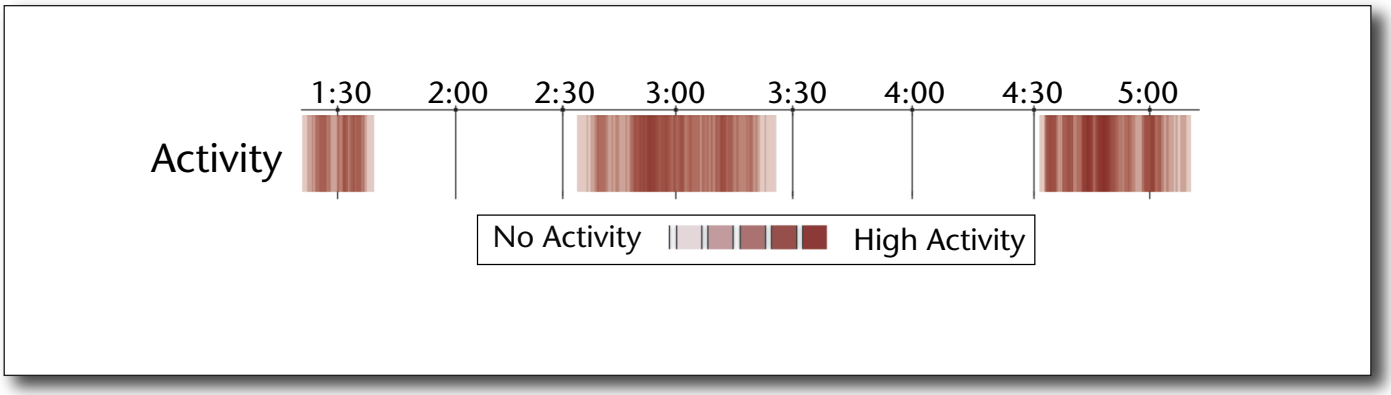

Figure 5. Example User History of a Typical User Showing Distinct Sessions of Activity with Breaks in Activity in Between.

Red lines indicate the intensity of user activity (darker red means more activity) and the white areas indicate lack of activity.

gaps between them, which splits this user's command history into three sessions.

Our command data set includes the time intervals between every pair of sequential commands performed, resulting in 40 million command pair timings. This data shows that 95 percent of the command intervals are shorter than one minute and 0.6 percent 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:

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

where c_i is the ith 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.

In the first axis of the design space (horizontal axis in figure 4), 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.

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 that was purchased recently by a user should have a greater impact on the prediction of the user's future behavior than an item that 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 has 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.

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 that 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 three-dimensional Alignment and two-dimensional Alignment commands, even though the user is currently working in a two-dimensional drafting project when clearly the two-dimensional 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.

The second axis of the design space (vertical axis in figure 4) determines how we generate command-by-command correlations. We split up the community database by either sessions or users.

Organization of Rating Database by Sessions (A and B)

Session-based correlation represents the similarity between two commands that 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 that would be seen over time periods spanning more than one session cannot be captured by using a session-based correlation method.

Organization of Rating Database by Users (C and D)

In contrast, to capture correlations that can occur

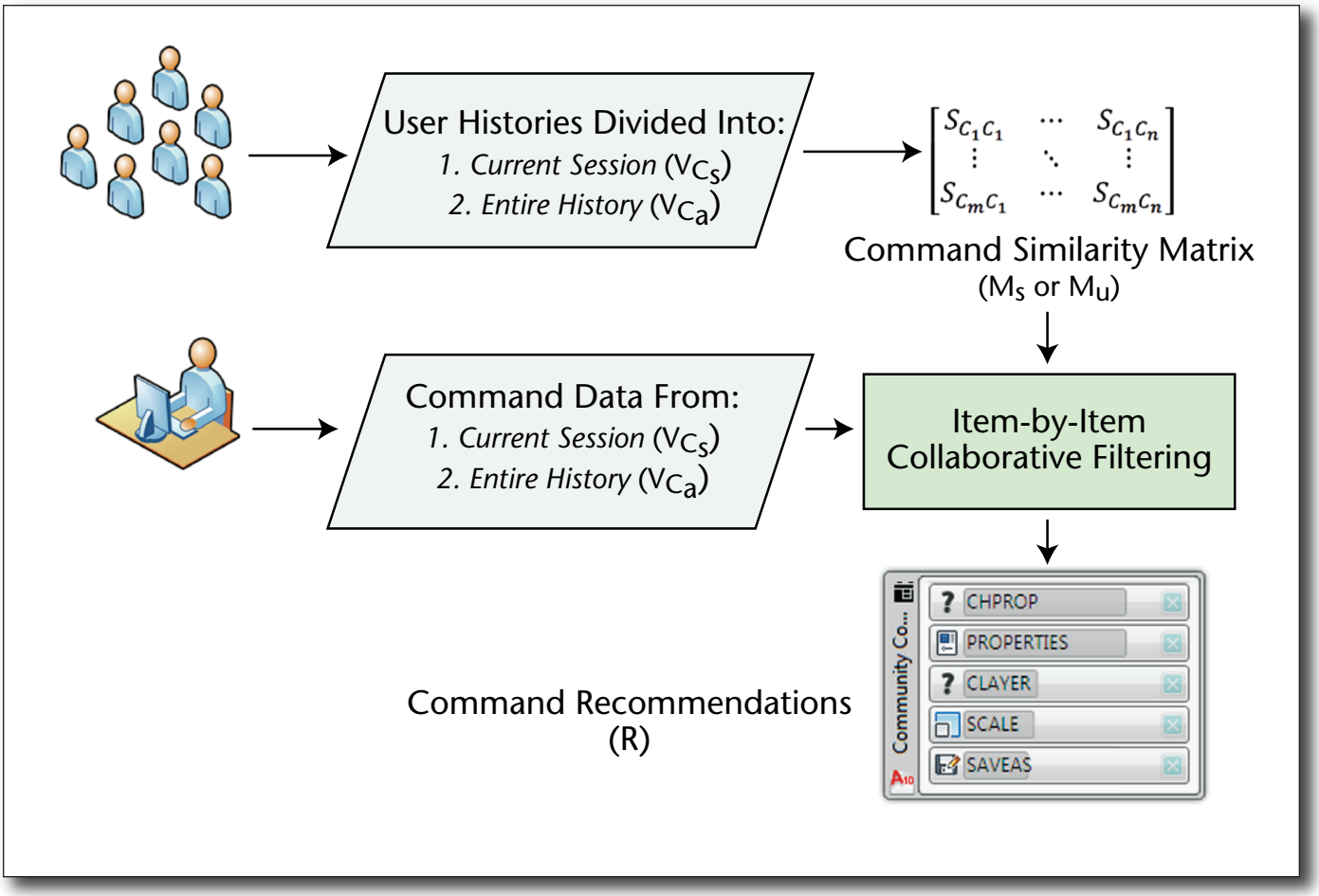

Figure 6. The Recommendation Process with the Two Decision Points of Using User Histories Broken Up into Sessions or Users.

Looking at the command data from the current session or the entire history of the active user.

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 figure 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 that may be used across different sessions.

We explored four item-based collaborative filtering algorithms, each designed to satisfy one of the four quadrants in figure 4.

A recommendation set *R* can then be generated for each of the four design quadrants from figure 4. Figure 6 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.

Offline 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 that dynamically measures the usefulness of suggested commands based on the sequential information in a user's command log (Matejka et al. 2009).

Consider a user u_i with a series of commands *S*. The *k*-tail evaluation divides this command sequence into a training sequence S_{train} and a testing sequence S_{test}, so that there are *k* unique commands in S_{train} that are not in S_{test}. For example, the command sequence in

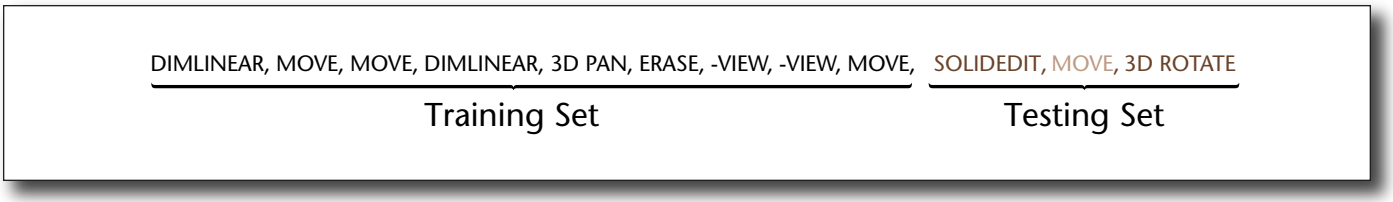

Figure 7. k-Tail Evaluation of a Command Sequence.

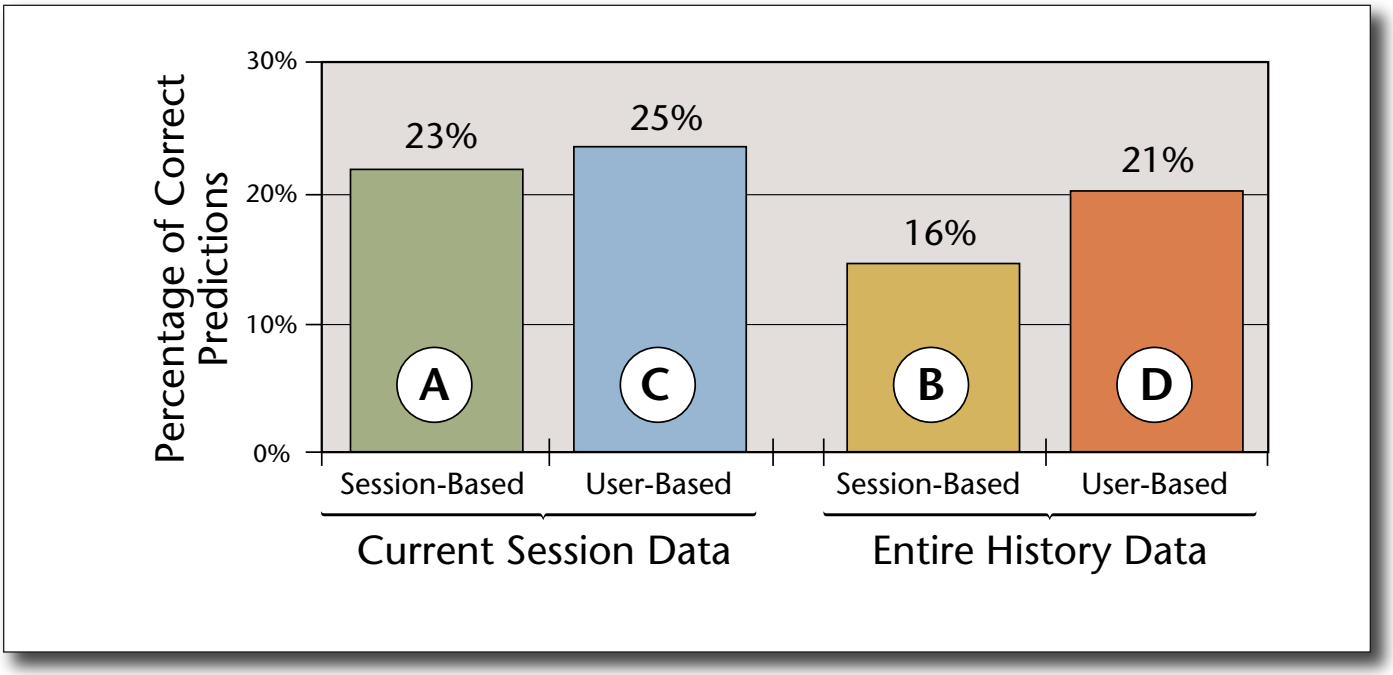

Figure 8. Offline Evaluation of Four Approaches in Figure 5.

figure 7 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 that are in both a user *i*'s recommendation list R_i, and the user's testing set $S_{test,i}$. We define the evaluation result of *k*-tail as hit_k, where *n* is the number of users in the community.

$$hit_k = \frac{\sum_{i=1}^{n} |R_i \cap ST_i|}{n}$$

We use the *k*-tail technique (with $k = 1$) to evaluate the recommendations generated from algorithms A through D (Li et al. 2011). Figure 8 shows that a user's more recent commands (A, C) seem to hold more relevance than commands that a user used any time in his/her entire 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.

We also propose to approximate a command recommendation's novelty factor using its binomial probability. We call this the *binomial novelty indicator* (BNI) (Li et al. 2011).

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

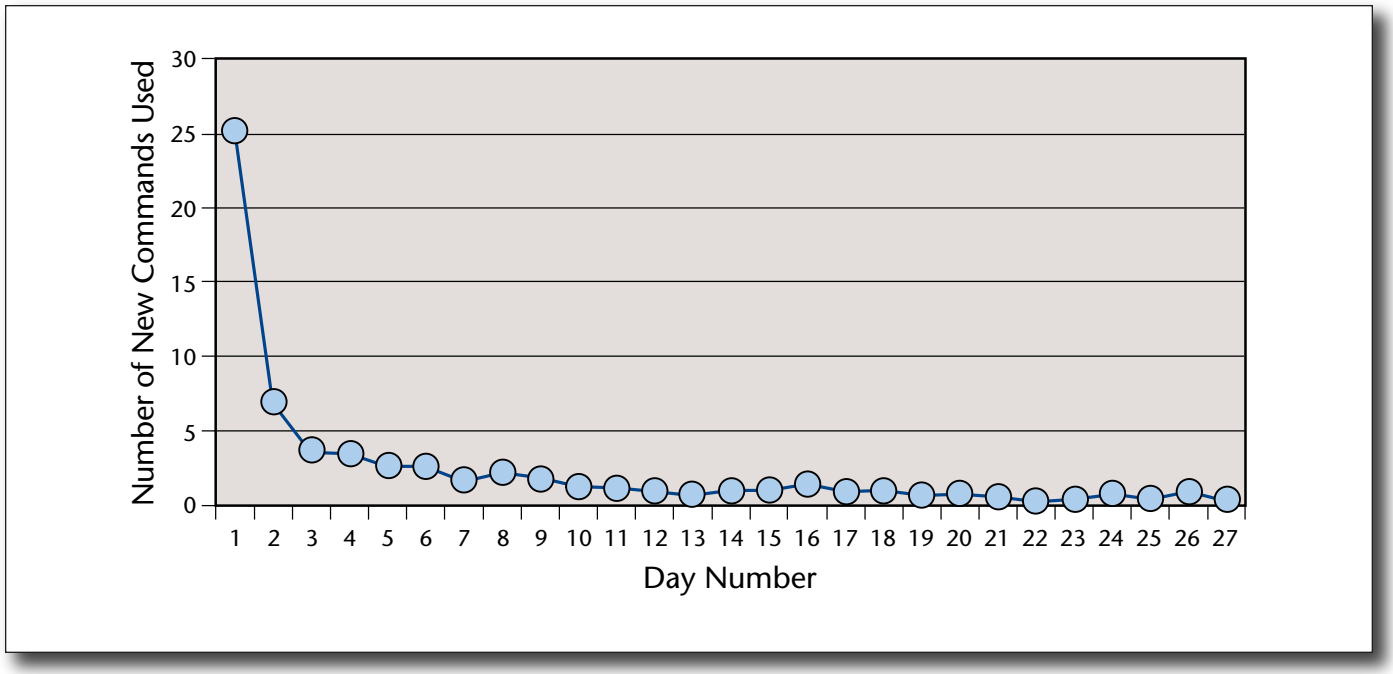

Figure 9. New Command Adoption Rates Based on 27 Users.

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

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 data set. 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

$$hit_k = \frac{\sum_{i=1}^{n} |R_i \cap ST_i|}{n}$$

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 13,000 commands. We compute that there is a 95 percent chance that A appears in the testing set once or more, and a 3 percent 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.

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 9). The data showed that the rate of using new commands levels off quickly. For example, after 8 days, 50 percent 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.

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 10).

In-Product Recommendations

Recommended commands are placed in a list within the AutoCAD plug-in palette (figure 11).

Clicking 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 12).

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 recom-

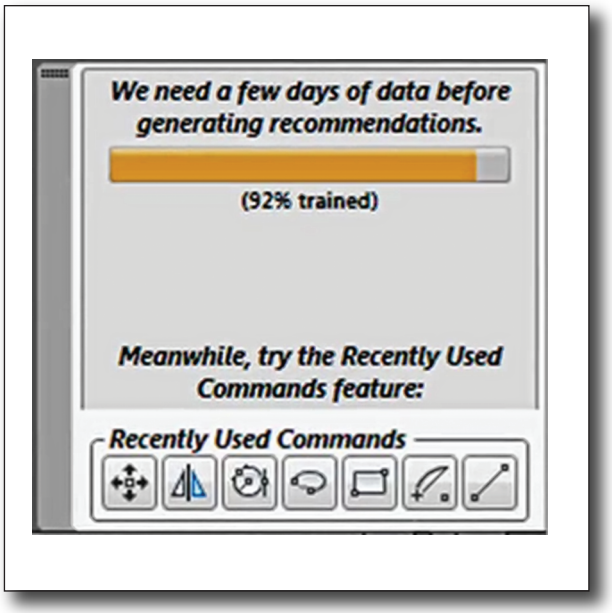

Figure 10. Recommender Training Phase UI.

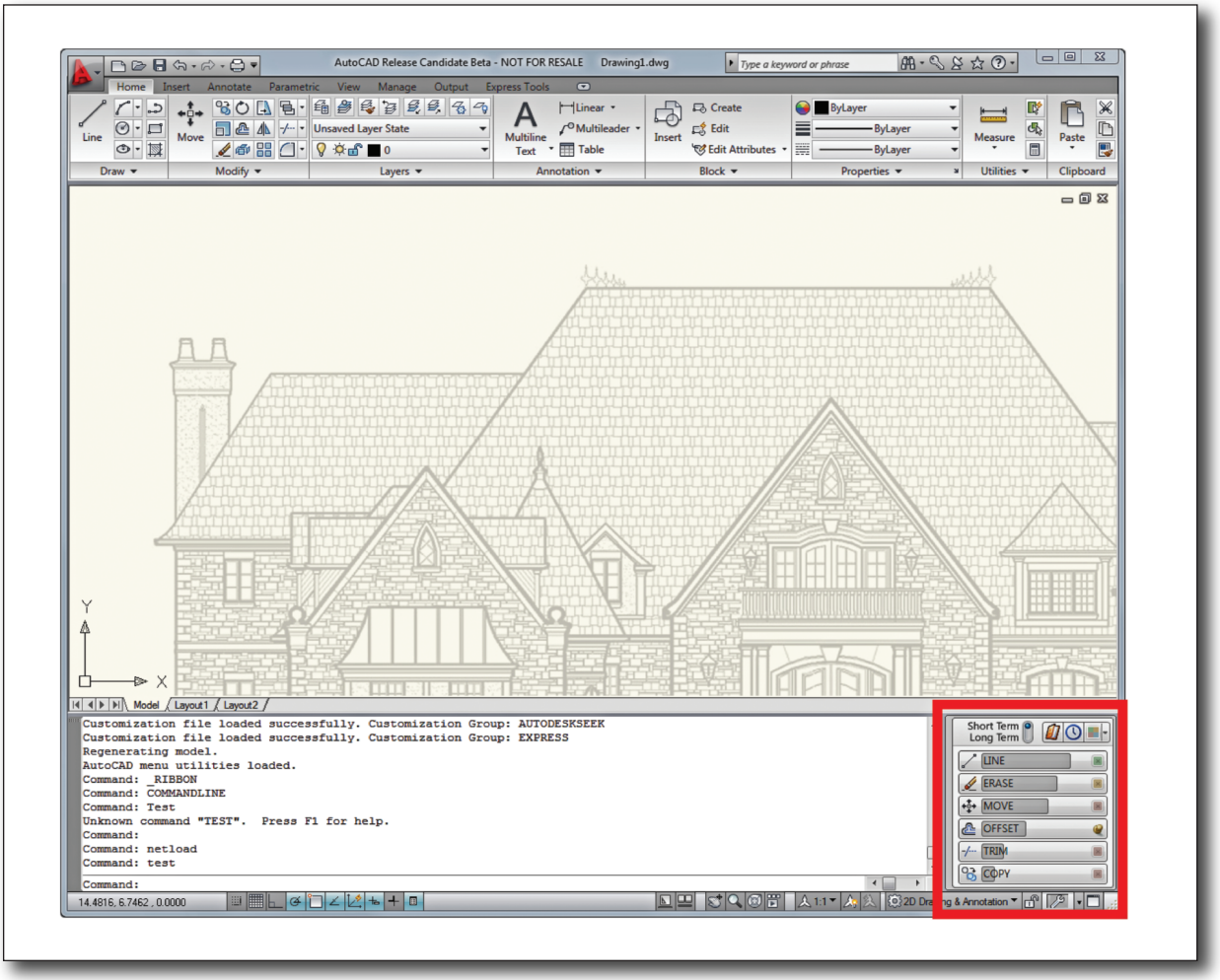

Figure 11. Recommender Plug-In Palette Is Opened in AutoCAD.

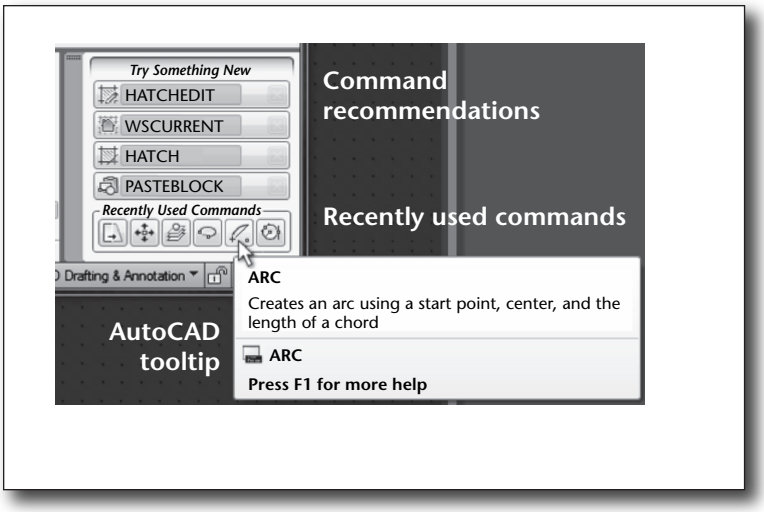

Figure 12. Recommended and Recently Used Commands.

Tooltip appears when mouse is hovered over the command.

mendations 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.

Application Use and Payoff

We report how long the CommunityCommands plug-in 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, and 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 13 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, 3 times, 10 times and 20 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 3 times, 9.6 for 10 times, and 7.3 for 20 times.

Figure 14 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 percent recommendation adoptions happened during the first 19 percent 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 percent) who downloaded the CommunityCommands plug-in enrolled in CIP. This of course means that 28.7 percent 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 that 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 of commands or relatively few.

Figure 15 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 shown 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 16 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 diagrams to the end users, to encourage usage reflection and further command adoption.

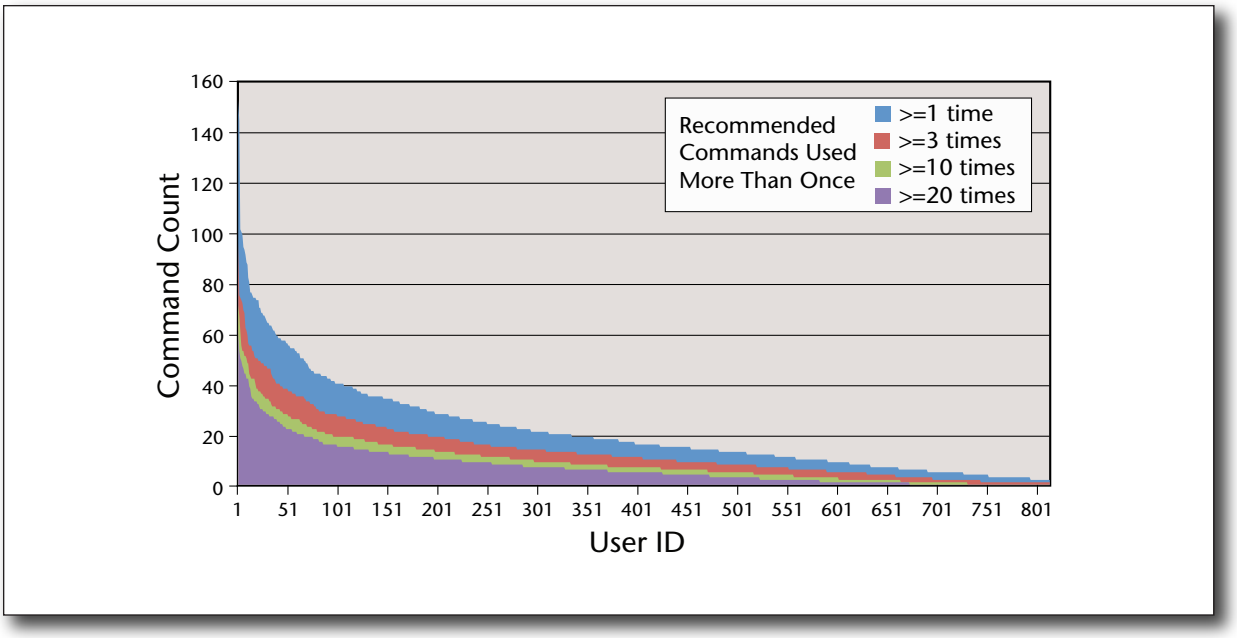

Figure 13. Recommended Command Adoption.

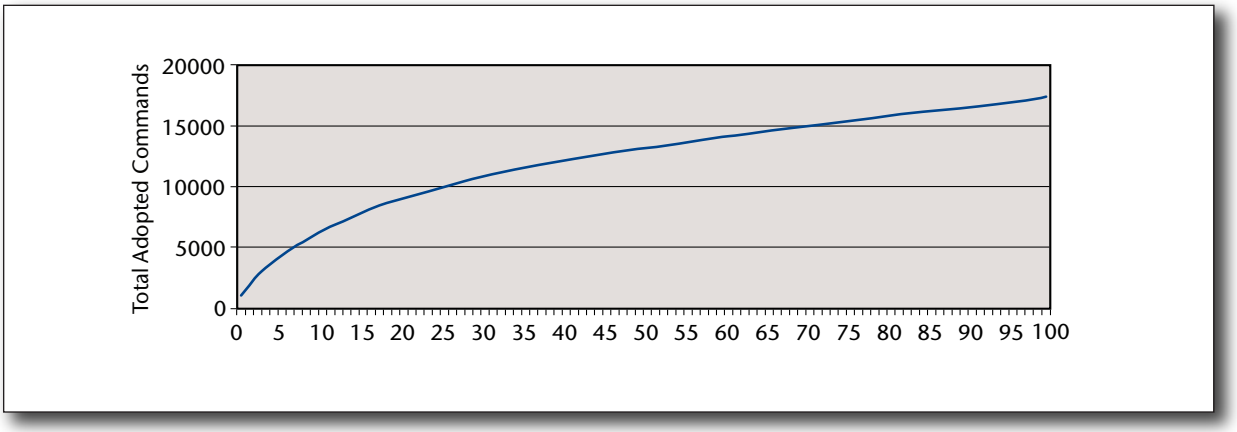

Figure 14. Total Adopted Useful Recommendations Over Deployed Time.

The horizontal axis shows the percentage of time passed.

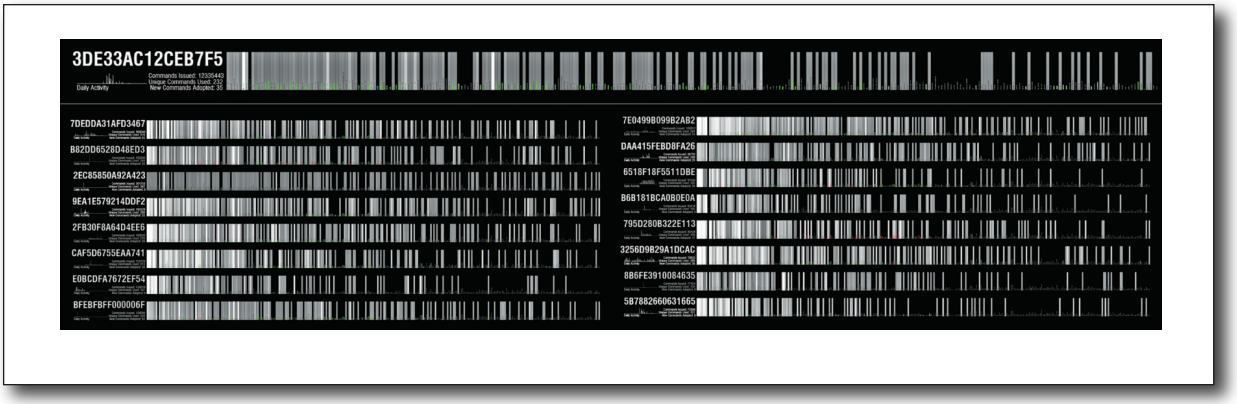

Figure 15. Legend of Software Usage DNA Diagram.

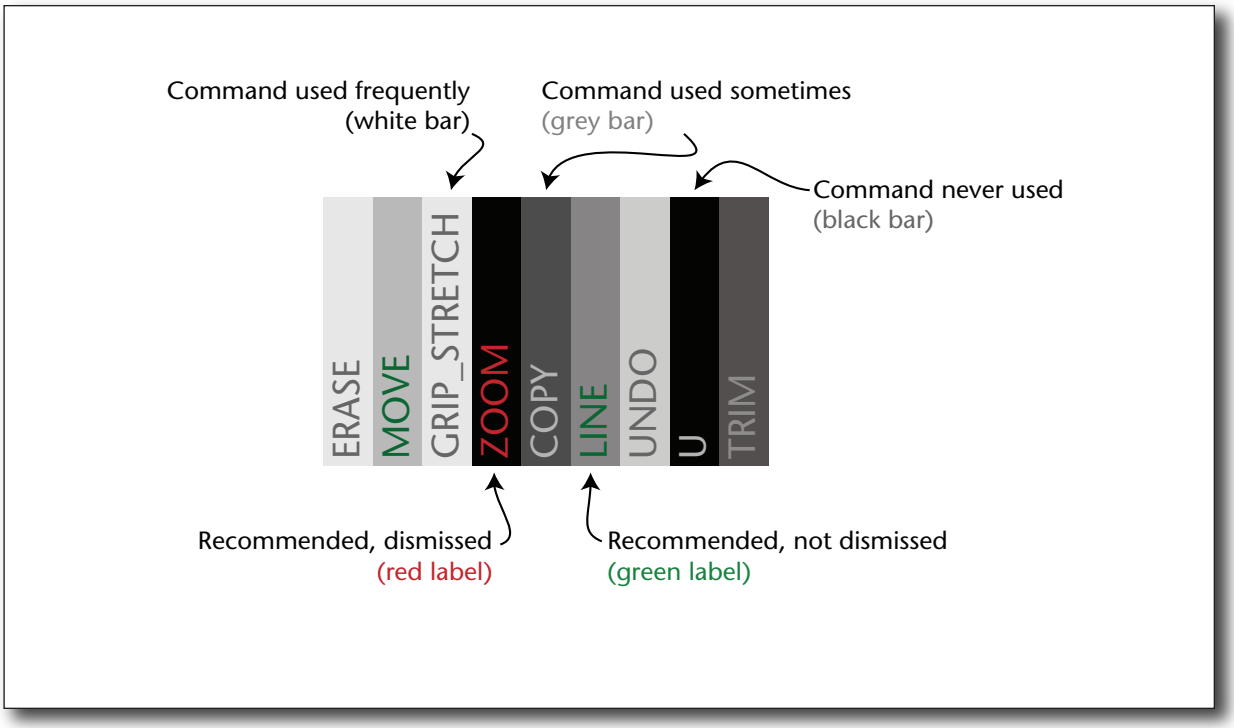

Figure 16. Software Usage DNA Diagrams from the 17 Most Active Users.

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 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 that 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.

Notes

1. www.autodesk.com.
2. ww.microsoft.com/products/ceip/EN-US/default.mspx.
3. www.autodesk.com/acip/CIP_Privacy_eng.html.

References

Ahmad, W., and Khokhar, A. 2007. An Architecture for Privacy Preserving Collaborative Filtering on Web Portals. In *Proceedings of the Third International Symposium on Information Assurance and Security,* 273–278. Plymouth, UK: University of Plymouth.

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 ACM Conference on Universal Usability — 2000,* 17–23. New York: Association for Computing Machinery.

Berkovsky, S.; Eytani, Y.; Kuflik, T.; and Ricci, F. 2007. Enhancing Privacy and Preserving Accuracy of a Distributed Collaborative Filtering. In Proceedings of the 2007 ACM Conference on Recommender Systems, 9–16. New York: Association for Computing Machinery.

Ding, Y., and Li, X. 2005. Time Weight Collaborative Filtering. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management,* 485–492. New York: Association for Computing Machinery.

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. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 565–572. New York: Association for Computing Machinery.

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*. New York: Association for Computing Machinery.

Herlocker, J. L.; Konstan, J. A.; Terveen, L. G.; and Riedl, J. T. 2004. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems* 22(1): 5–53.

Hill, W.; Stead, L.; Rosenstein, M.; and Furnas, G. 1995. Recommending and Evaluating Choices in a Virtual Community of Use. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, 194–201. New York: Association for Computing Machinery.

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. In *Proceedings of the Tenth ACM International Conference on Information and Knowledge Management*. 247–254. New York: Association for Computing Machinery.

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 Transactions Computer-Human Interaction,* 18(2): 6: 1–6: 35.

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. In *Proceedings of the 22nd ACM Symposium on User Interface Software and Technology,* 193–202. New York: Association for Computing Machinery.

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*. 263–266. New York: Association for Computing Machinery.

Mitchell, J., and Shneiderman, B. 1989. Dynamic Versus Static Menus: An Exploratory Comparison. *SIGCHI Bulletin* 20(4): 33–37.

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

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. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, 175–186. New York: Association for Computing Machinery.

Schein, A.; Popescul, A.; Ungar, L.; and Pennock, D. 2001. Generative Models for Cold-Start Recommendations. Paper presented at the 2001 ACM SIGIR Workshop on Recommender Systems, September, New Orleans, LA.

Shardanand, U., and Maes, P. 1995. Social Information Filtering: Algorithms for Automating Word of Mouth. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* 210–217. New York: Association for Computing Machinery.

Shneiderman, B. 1983. Direct Manipulation: A Step Beyond Programming Languages. *IEEE 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. In *Proceedings of the Third ACM Conference on Recommender Systems,* 157–164. New York: Association for Computing Machinery.

Twidale, M. B. (2005). Over the Shoulder Learning: Supporting Brief Informal Learning. *Computer Supported Cooperative Work* 14(6): 505–547.

Wei Li, Ph.D., is a principal research scientist at Autodesk Research. Li's main research interests are interdisciplinary, drawing on both artificial intelligence and human-computer interaction. In particular, his past and present work/interests fall into the following areas: general learning mechanisms in software applications, gamification, recommender systems, knowledge representations, and automatic reasoning. Through a mixture of empirical work and systems building, Li currently focuses on novel techniques for goal-directed design systems and data-aided design.

Tovi Grossman, Ph.D., is a senior principal research scientist at Autodesk Research, located in downtown Toronto.

His research is in human-computer interaction, focused on understanding and improving software learnability in complex end user applications. Although this is one of the most fundamental problems of human-computer interaction, new trends and technologies allow us to think about the problem in exciting new ways that were not possible in the early days of human-computer interaction research. His other research passion is interaction techniques, and in particular, for new technologies, such as multitouch, miniature projectors, and three-dimensional displays.

Justin Matejka is a senior research scientist at Autodesk Research in Toronto, Ontario. He has been with Autodesk for more than eight years researching user interfaces for three-dimensional navigation, developing new techniques for navigating streaming videos, and developing systems to improve the learnability of complex software applications. His current research focus is in creating interactive visualization techniques for making sense of large data sets.

George Fitzmaurice, Ph.D., is a director of research and runs the User Interface Research Group for Autodesk. He has been with Autodesk (including Alias) for more than 15 years conducting research in two-dimensional and three-dimensional user interfaces (UIs) including input devices, large displays, two-handed interaction, multitouch, pen-based UIs, tracking menus, spatially aware displays, three-dimensional navigation, and tangible UIs. In the human-computer interaction community he established the field of graspable UIs, which is the precursor to what is known as tangible UIs. Currently, he is leading research projects on advanced learning technologies for feature-rich software applications.