Educational Technology & Society 3(1) 2000
ISSN 1436-4522

OWL: A Recommender System for Organization-Wide Learning

Frank Linton, Ed.D.
Lead Scientist
Mailstop K302, The MITRE Corporation
202 Burlington Road, Bedford MA 01730 USA
Tel: +1-781-271-2877
Fax: +1-781-271-2352
linton@mitre.org

Deborah Joy
Internet Applications Technician, The MITRE Corporation

Hans-Peter Schaefer
Senior Artificial Intelligence Engineer, The MITRE Corporation

Andrew Charron
Senior Information Systems Engineer, The MITRE Corporation

ABSTRACT

We describe the use of a recommender system to enable continuous knowledge acquisition and individualized tutoring of application software across an organization. Installing such systems will result in the capture of evolving expertise and in organization-wide learning (OWL). We present the results of a year-long naturalistic inquiry into an application's usage patterns, based on logging users' actions. We analyze the data to develop user models, individualized expert models, and instructional indicators. We show how this information is used to recommend learning tips to users.

Keywords

Recommender system, organization-wide learning, OWL, individualized instruction, agent, instrumentation, logging.

Introduction

New Workplace Technologies Enable New Learning Methods

In the last decade, an enormous change has taken place in the workplace: there is a PC on every desk, and much office work is performed in the medium of software. Mastering one's software - at least the portion of it that is relevant to one's job tasks - has become a key enabling skill (a propadeutic skill) for performing well on the job. It also happens that work performed in software is easily observable by automated processes. This observability provides new opportunities for training, provided the software is *instrumented*.

Software is instrumented by making small code changes in the software, or by adding code to the computing environment, that observes and reports actions users take on the software's interface. Thus, the user's every menu selection, button click, and keyboard command are observed and reported.

This new knowledge-capture method - instrumentation - opens the door to new training methods. These methods include *embedded* training, where the training for an application appears, from the user's perspective, as an integral part of the application; *distance* learning, where the learning takes place at the user's workplace, rather than in a classroom; *intelligent* tutoring, where software observes the user doing realistic job tasks (but doing them in a training context) and provides coaching and feedback; and *collaborative* learning, where students share the use of software tools. Together then, instrumentation enables EDICT, embedded, distributed, intelligent, collaborative training. Tools that build on these methods include Intelligent Tutoring Systems, Self-checking Checklists, Peer Help Systems (McCalla et.al, 1997), and Recommender Systems (Resnick & Varian, 1997).

At MITRE, we have developed software tools to instrument user actions in, UNIX, Java, and Microsoft Office. These tools are freely available to anyone and may be obtained by contacting the first author.

Instrumentation can serve both to capture experts' actions on the job and to capture learners' actions during training. Logging captures data about *how* the work is done, automatically, in great detail, and from a large

number of experts. When used in an instructional mode instrumentation captures student actions, which can then be interpreted in various ways using the EDICT methods mentioned above. In OWL (for Organization-Wide Learning), the software we report on here, we combine knowledge capture and instruction in one integrated process, capturing expertise from anyone who displays it and delivering instruction to everyone who needs it. The distinction between expert and novice blurs, as individuals prove to be experts in some areas and novices in others.

Recommender Systems

OWL is a kind of recommender system (Resnick & Varian, 1997). Recommender systems are frequently used in electronic commerce to recommend purchases to consumers. In contrast, we use a recommender system to recommend learning to application users. There are several types of recommender system, we review each briefly.

One type of recommender system is found in Information Retrieval (IR) systems, it recommends 'other documents like this one.' In other words, it recommends documents that share many attributes with a document the user identifies as desirable.

Another type of recommender system is found in targeted marketing, it recommends purchases that 'people similar to you' like. In other words, it recommends purchases found to be desirable by people who share many attributes of your demographic profile.

OWL uses a third type of recommender system, as do the e-commerce web sites such as Reel.com. It is based on neither the products' nor the consumers' attributes, but on individuals' patterns of purchases (or in OWL's case, the software users' preferential selections of functionality). For example, consider the group of people who like the movies you like and dislike the movies you dislike. Some members of the group will have undoubtedly seen and liked movies that you have not yet seen. The Reel.com recommender system would suggest those movies to you. Similarly, OWL compares each software user to others to find those who share patterns of command usage, then recommends new commands for each user to learn based on the pooled knowledge of similar users.

In the relatively unstructured world of real work, it is very difficult to infer what a user is attempting to do in a given situation with enough certainty to recommend the best actions for them to take, although Microsoft's Office Assistant represents a valiant attempt in this direction. In contrast, in OWL we assume that, if a user knows several ways to, for example, delete text, they will choose the appropriate one for each occasion. From observation of a pool of users, OWL determines the relative utility of each command, and OWL introduces them to the individual users over time, as needed by the individual, and according to their relative utility, so the user is always learning the next most useful functionality. These issues are considered further in the Discussion section.

The functionality of a modern application, such as Microsoft Word, may consist of more than 1000 distinct commands. The majority of those commands are irrelevant to the job tasks of any one user, so any attempt to master the full functionality of an application is counter-productive. On the other hand, most users learn only a small part of the relevant functionality of an application in formal training. Knowledge of the remaining relevant functionality is acquired, if it is acquired at all, slowly and sometimes painfully over the years by trial and error, in conversations with peers, etc. Meanwhile, the lack of knowledge results in reduced effectiveness and efficiency. A recommender system will help users to acquire the relevant functionality of the application quickly by suggesting to each user functionality that their peers have already found useful.

> "In many workplaces ... mastery is in short supply and what is required is a kind of collaborative bootstrapping of expertise."
>
> (Eales & Welch, 1995, p. 100)

Recommender Systems for Learning

OWL has been gathering data from users for several years, and we have recently implemented the software that makes recommendations to OWL users. With the empirical evidence not yet in, we cannot claim in this report that the OWL recommender system successfully promotes new learning, but there are several reasons to expect it will. First, we know that recommender systems for e-commerce work. Recommender systems are acknowledged to account for a large part of Amazon.com's success and are beginning to cause a profound change in the

marketing of so-called taste goods (Gladwell, 1999). Second, the recommendations are timely. Information is presented to the user at the time that is most appropriate for her to learn it, when it fills a gap in her knowledge or extends the boundaries of her knowledge.

Third, we know that the functionality OWL is recommending is relevant, because the learner's peers in the same job context are already using it. Most software applications have a large number of functions that are not applicable to the job tasks performed by any one individual. For example, we will present data that show only about 15% of Word's commands are applied by the users OWL monitors. Learning the never-used functions would be a waste of effort. On the other hand, most individuals learn only a small portion of the useful functionality of an application during formal training. Thus the individual is left to her own devices to determine and to acquire most of the productivity-enhancing functions of the applications on her desktop.

Fourth, we expect the recommendations to be time-saving. To learn more about an application, an individual can turn to advanced courses, self-study and reference materials, peers, user groups, the world-wide web, etc. Nevertheless, the user faces a difficult task. She must not only perceive a need but also perceive the software as having the possibility of fulfilling it. She must then explore the software to find the function or set of functions that enables her to do the task with the software. Because the user and the software developer often define the same task in different terms, such explorations are often fruitless, and are correctly seen as high-risk, low-return ventures. It should come as no surprise then, that any new learning resulting from these perceptions and explorations is rare, and that as a result, much of the productivity gain made possible by desktop software is never realized (Landauer, 1996).

OWL provides a solution to this problem. OWL pools the knowledge of all users in an organization - by recording each use of each function. Each use of a function is interpreted as a vote for its utility. OWL then compares the functions used by each individual to the functions used by the pooled group of users. It finds the gaps in and the boundaries of each individual's knowledge, and makes individualized learning recommendations to fill in the gaps and extend the boundaries of their knowledge. OWL provides a current, peer-based answer to the question: Given all the functionality of this application, which is the next-most useful function for me to learn? OWL provides users with a means of learning that is better than exploring but OWL does not discourage or inhibit exploration, and when a user discovers something and begins to use it, it is added to OWL's database and will, at the appropriate time, be recommended to each of the other users.

In the remainder of this paper we describe the logging process and make some initial remarks about modeling and coaching software users. We then present an analysis of the data we have logged and our process of creating individual models of expertise. Next we show how individualized coaching and feedback is presented to users. Then we discuss several issues raised by the analysis and make a few closing remarks.

OWL Research

This research explores the potential of a new sort of user modeling based on summaries of logged user data. This method of user modeling enables the observation of a large number of users over a long period of time, enables concurrent development of student models and individualized expert models, and applies recommender system techniques to on-the-job instruction. Earlier work is reported in Linton (1990) and Linton (1996). Kay and Thomas (1995) and Thomas (1996, 1998) report on related work with a text editor in an academic environment.

OWL (1) captures evolving expertise from a community of practice (Lave & Wenger 1991), (2) supports less-skilled members of the community in acquiring that expertise, and (3) serves as an organizational memory for the expertise it captures.

The main goal of the approach taken in this work is to continuously improve the performance of application users. The system described here would be applicable in any situation where a number of application users perform similar tasks on networked computers.

The OWL Logger

From a practical standpoint, it is crucial that the logging process be reliable, unobtrusive, and frugal with resources, as observation takes place for extended periods of time (Kay & Thomas, 1995). The research reported here is based on logs of Microsoft Word users. The OWL logger was written in Visual Basic for Applications. In

general, it is difficult to implement loggers without access to the application's source code, but Cheikes, et.al. (1998) make available a tool for instrumenting UNIX applications. A similar tool is available for applications written in Java. As mentioned, the OWL, UNIX, and Java loggers may be obtained by contacting the first author.

The OWL logger works as follows. Each time a user issues a Word command such as Cut or Paste, the command is written to the log, together with a time stamp, and then executed. Logging is initiated when the user opens Word; a separate log is created for each file the user edits, and when the user quits Word, the logs are sent to a server where they are periodically loaded into a database for analysis. A toolbar button labeled 'OWL is ON' (or OFF) informs users of OWL's state and gives them full control of the logging process.

The first few lines of the log record general information: the logger version, the date and time stamp, and the author; followed by the platform, processor, and version of Word. Then detailed logging begins. Each time the user enters a Word command, the logger adds a line to the log file. Each line contains a time stamp, the command name, and possibly one or more arguments. For example, one log entry recorded these facts: at 5:11:34 p.m. the author used the FileOpen command to open the file entitled Notes for ITS 98. Other entries record that the author performed some minor editing (copy, paste, etc.), and then printed the file. The log does not record text a user enters; this omits some potentially useful information but preserves users' privacy and makes logging more acceptable.

Logging captures a detailed record of a user's activities but the record may be sketchy for at least two reasons. First, an individual may also edit text on other systems without loggers, so some of their activity may not be captured. Second, a macro-based logger besides omitting text, does not capture certain other keyboard actions such as Tab and Return, nor does it capture certain mouse actions such as scrolling, nor does it distinguish *how* commands are entered. Finally, permitting user control over logging means that logging can be turned on and off at will, though the default is that OWL is on. To summarize then, the logged data is not a census of the user's actions, nor a random sample, but rather an arbitrary selection of them.

Modeling and Coaching Users

In the Analysis section we will analyze and synthesize the software usage patterns of a group of users to create an expert model for each individual and show how to use that model for coaching. One benefit of the recommender system approach is that there is no need to capture expertise before installing the system. Whenever someone begins to use a new command, the event is captured and eventually the command is recommended to others. We define the characteristics of a model of expertise as the number and relative frequency of commands used. Because individuals' use of commands will vary, models of expertise are created by observing and interpreting data from multiple individuals.

The system is intended to continuously improve the performance of application users by providing individualized coaching, either in the form of OWL Tips or via a Skill-O-Meter (see Providing Feedback to Users). The coaching is based on a comparison of a user's logged data to an expert model. The Tip system compares the individual user's model to the pooled performance of their peers, and makes recommendations for learning. The system attempts to time the recommendation such that the user already will have felt a need for the knowledge the system is offering to teach ("There must be a better way to do this, but I don't have time to look for it now"). Besides tips, the system will provide an interface that will permit users to explore the relationship between their current and expert models, a Skill-O-Meter.

Applications are frequently upgraded, work processes are refined, and clever users discover better ways of doing things, thus expertise is not static, but rather, it evolves. To capture evolving expertise, the system must update its expert models monthly. The updating process begins by analyzing user data. The Analysis section presents the method we use for detecting the anomalous use of commands. Some users are experts and generally innovative practitioners, yet any user may be the first to apply a command. As users discover and apply new commands these will appear in their user models. Other users' expert models will eventually reflect the existence of each new command, the recommender mechanism will recommend it, and the knowledge will spread.

This section has described the characteristics of a recommender system for coaching software skills on the job, based on logged user data. The next section will present an analysis of data that has been collected over a one-year period.

Analysis

This section presents an analysis of log data and shows how it is used for tutoring learners incrementally to improve their software skills. First we present some summary statistics of the users and their log data and show how the volume of data logged influences the appearance of expertise. Next we describe the relative frequencies of the different types of commands. We then present a table showing the contribution of each command to the total, and give an equation characterizing the relationship. We then consider the implications of these facts for user modeling and describe a means of being reasonably certain that we have examined sufficient log data to have observed a command in use. Finally we describe a way to derive users' expected behavior from their observed behavior and then, by comparing and contrasting these behaviors, derive *instructional indicators*, which, when taken together with other knowledge, OWL uses to provide learning guidance to users.

The analysis presented here is exploratory in nature. The method we have used is Naturalistic Inquiry, which, to paraphrase Patton (1990, p. 40, 41) involves studying real-world situations as they unfold naturally in a non-manipulative, unobtrusive, and non-controlling manner, with openness to whatever emerges and a lack of predetermined constraints on outcomes. The point is to understand naturally occurring phenomena in their naturally occurring states.

Software Users

The project obtained substantive logs from 16 users. The majority of them were members of one section of The MITRE Corporation's Advanced Information Systems Center. MITRE is a federally funded not-for-profit corporation doing research in the public interest. The users consisted of one group leader, ten Artificial Intelligence Engineers, three technical staff, and two support staff. There were eight males and eight females. The users had been employed at MITRE from one to twenty-nine years with a median of eight years. The users worked on four different Macintosh platforms, three versions of the Macintosh Operating System, and three versions of Word 6.0 for the Macintosh. The period of logging ranged from 3 to 11 months per person. The project acquired a total of 96 user-months of data. During the time they were logged, the users -- as a group -- applied 152 of the 642 available Word commands a total of 39,321 times.

The average person used 56 different commands in the period they were logged (average of 6 months); applying 25 different commands 409 times in the average month.

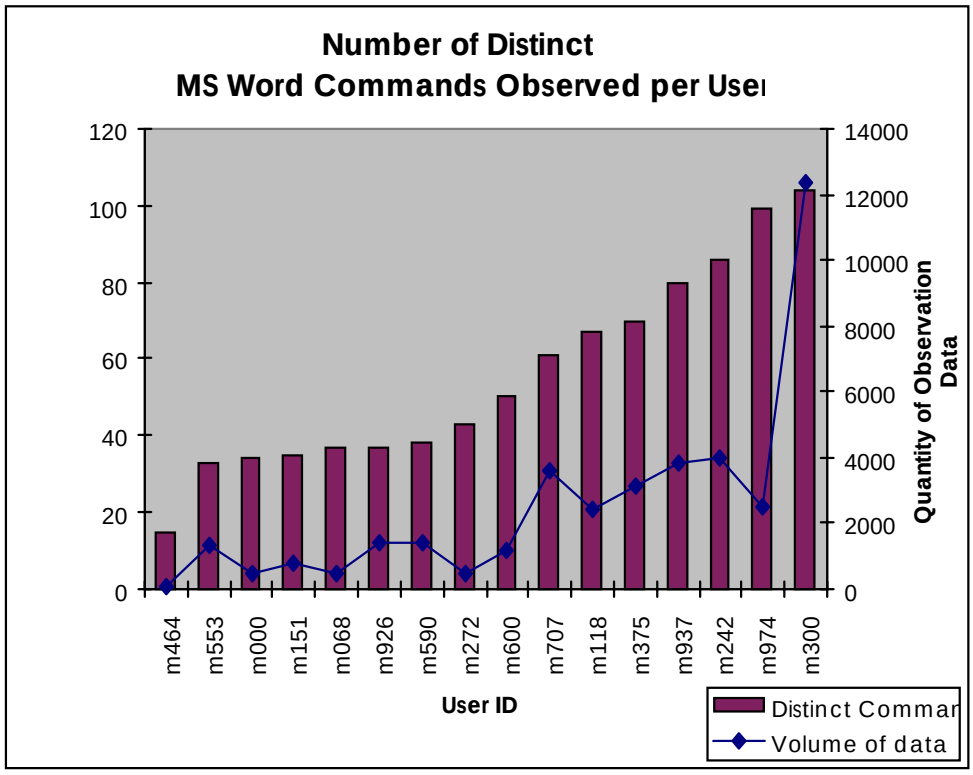

Figure 1. User Data

As Figure 1 shows, the apparent skill level of users varied by a factor of 7: from 15 to 104 commands. Even ignoring the extremes of performance, it is evident that some users are much more skilled than others and there is ample opportunity for further learning.

However, further inspection of Figure 1 reveals a complication: an individual's skill level is correlated (0.776) with the amount of observation data; the more a person has been observed, the more she apparently knows. This phenomenon appears because many commands are rarely used, and observation must continue for a long time before the command is observed. Thus it is not sufficient to observe an individual for some predetermined period of time (or predetermined number of log entries) to assess her knowledge; extra effort is be required to determine skill level from a finite observation sample, and to disambiguate the effects of observation from learning.

Command Usage

We turn now to a discussion of how commands are used. We begin with an overall, descriptive view. One of the most salient characteristics of the recorded data is the relative inequality in the use of each type of command. For example, as shown in Figure 2 Command Type Usage, the File commands, i.e., the various commands under 'File' in Word's main menu, make up nearly 48% of all commands used, while the various Help commands account for only 0.09 % of the commands logged.

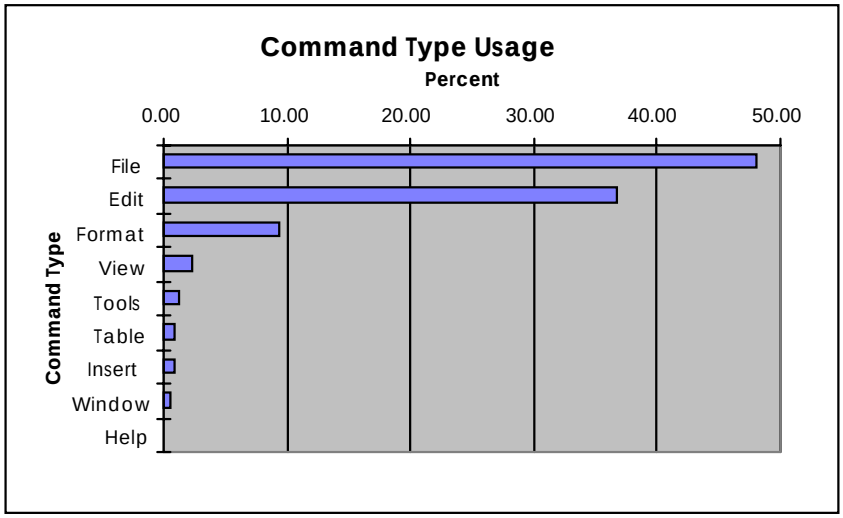

Figure 2. Command Type Usage

Table 1 lists the 20 most frequently occurring Word commands sequenced by the frequency of their occurrence, together with the percentage occurrence of each, and the cumulative percent of usage for all commands up to that point in the sequence. Command names are preceded by their main menu type; e.g., FileOpen is the Open command on the File menu. The first 10 commands account for 80%, the first 20 commands account for 90%, etc.

The chart in Figure 3 presents command usage data for the 100 most-frequently-used Word commands. The horizontal axis represents commands 1 through 100 (the names of the first 20 of these commands were itemized in Table 1). Each command's usage is indicated by the Percent line relating to the logarithmic scale on the left margin of the chart. Note that command usage (when expressed in percent) varies by more than three orders of magnitude. The Trendline plotted, which most-closely fits the observed data ($R^2 = 0.96$), is a power curve. An exponential curve also provides a close fit ($R^2 = 0.90$) but it badly misestimates the first 12 values (in contrast, the power curve misestimates only the first four values). The power curve equation describing Word command frequency of use is

$$y = 137\ x^{-1.88}$$

This equation is in contrast to the exponential distribution reported by Thomas (1996) for Sam editing commands, and the Zipf distribution reported by others (Thomas, 1998) for the UNIX domain.

Sequence	Command	Percent	Cumulative Percent
1	FileOpen	13.68	13.68
2	EditPaste	12.50	26.18
3	FileSave	11.03	37.22
4	FileDocClose	10.25	47.47
5	EditClear	9.50	56.97
6	EditCopy	7.86	64.83
7	FormatBold	4.22	69.05
8	FilePrint	4.12	73.16
9	EditCut	3.50	76.66
10	FileQuit	2.73	79.40
11	FileSaveAs	2.17	81.57
12	FilePrintDefault	1.23	82.81
13	EditUndo	1.16	83.97
14	FormatUnderline	0.94	84.90
15	FileNew	0.90	85.81
16	EditFind	0.85	86.66
17	FormatCenterPara	0.79	87.45
18	ToolsSpelling	0.75	88.19
19	FilePrintPreview	0.74	88.94
20	ViewHeader	0.68	89.62

Table 1. Command Sequences and Percentages

The line formed by the light-colored triangles in the chart in Figure 3 plots the cumulative percent of data against the axis on the right margin of the chart. As mentioned, relatively few commands account for the bulk of the commands used.

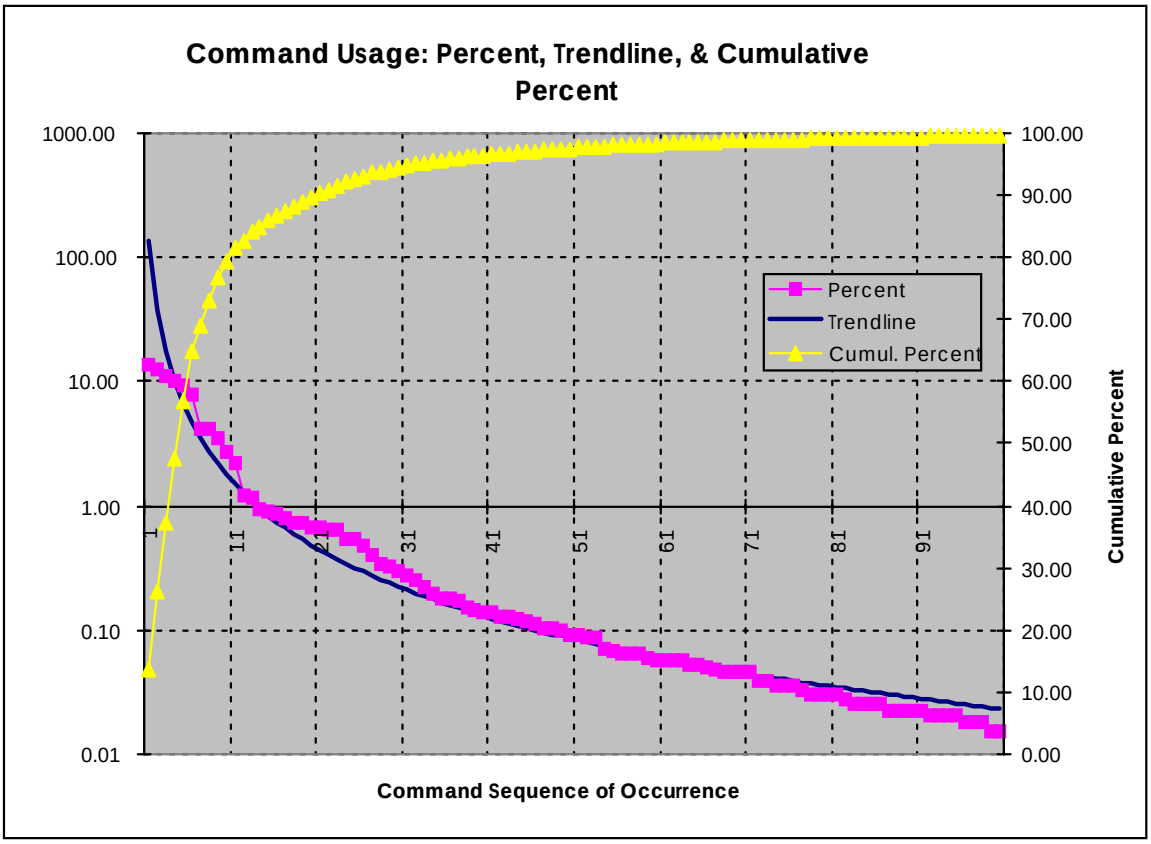

Figure 3. Command Trendline

Now that we have an overall view of the data, we turn to a specific question that must be answered if we are to provide learning guidance to users of software applications. We want to know, for a specific command and a given log length, whether we have enough data to be confident in drawing inferences about the user's knowledge. In particular, we are interested in the situation where we have no data from the user on a particular

command. How much data do we need in order to distinguish between the case where the user does not know the command, and the case were the individual knows the command but simply has not encountered an opportunity to use it?

We intuit that a data sample of a given size will contain a better representation of the more frequently occurring commands than of the rarely occurring ones; and we can quantify this intuition by applying the notion of 'confidence interval' as described in Triola (1983). It turns out, however, that to be quite certain that a user does not know a rarely used command is impractical; details are available in Linton (1999). OWL can be quite sure of its recommendations with respect to gaps in a user's knowledge, but may suggest the user learn commands they already know when making recommendations at the boundaries of a user's knowledge. Only empirical evaluation will determine an acceptable error rate. We note in passing, however, that spelling checkers are quite useful even though they mistakenly call many correctly spelled words to our attention.

Individual Models of Expertise

We illustrate the recommendation generation algorithm with the following analysis. For ease of illustration, we have limited our analysis to the commands appearing under Word's Edit sub-menu. OWL performs a similar analysis for every command. The first of the three tables in Figure 4 presents data for each of our 16 users. In the table, each column contains data for one user and each row contains data for one command (Edit commands that were not used have been omitted). A cell then, contains the count of the number of times the individual has used the command. The columns have been sorted so that the person using the most commands is on the left and the person using the fewest is on the right. Similarly, the rows have been sorted so that the most frequently used command is in the top row and the least frequently used command is in the bottom row. Consequently the cells with the largest values are in the upper left corner and those with the smallest values are in the lower right corner. The table has been shaded to make the contours of the numbers visible: the largest numbers have the darkest shading and the smallest numbers have no shading, each shade indicates an order of magnitude.

Inspection of the first table reveals that users tend to acquire the Edit commands in a specific sequence, i.e., those that know fewer commands know a subset of the commands used by their more-knowledgeable peers. If instead, users acquired commands in an idiosyncratic order, the data would not sort as it does. And if they acquired commands in a manner that strongly reflected their job tasks or their writing tasks, there would be subgroups of users who shared common commands. Readers may also observe that the more-knowledgeable users do not replace commands learned early on with more powerful commands, but instead keep adding new commands to their repertoire. Finally, the sequence of command acquisition corresponds to the commands' frequency of use. While this last point is not necessarily a surprise, neither is it a given.

There are some peaks and valleys in the data as sorted, and a fairly rough edge where commands transition from being used rarely to being used not at all. These peaks, valleys, and rough edges may represent periods of repetitive tasks or lack of data, respectively, or they may represent overdependence on some command that has a more powerful substitute or ignorance of a command or of a task (a sequence of commands) that uses the command. In other words, some of the peaks, valleys, and rough edges may represent opportunities to learn more effective use of the software.

In the second table in Figure 4 the data have been smoothed. The *observed* value in each cell has been replaced by an *expected* value, the most likely value for the cell, using a method based on the row, column and grand totals for the table (Howell, 1982). In the case of software use, the row effect is the overall relative utility of the command (for all users) and the column effect is the usage of related commands by the individual user. The expected value is the usage the command would have if the individual used it in a manner consistent with her usage of related commands and consistent with her peers' usage of the command.

These expected values are a new kind of *expert model*, one that is unique to each individual and each moment in time; the expected value in each cell reflects the individual's use of related commands, and their peers' use of the same command. The reason for differences between observed and expected values, between one's actual and expert model, might have several explanations such as the individual's tasks, preferences, experiences, or hardware, but for the purpose of making a recommendation, we assume the difference indicates the lack of knowledge or skill.

In the third table in Figure 4, each cell contains one of five symbols. The symbols are indicators of learning opportunities that are used by the recommendation generation process. These indicators are data that, combined

with domain and curriculum knowledge, result in recommendations for learning (OWL's Tips). The five symbols are: "_" (underscore), " " (blank), "New," "More," and "Alt." In the figure, "New" has the darkest shading, followed by "More" and "Alt." The underscore and blank have no shading.

		Observed														
Count of COMMAND		EMPNMBR														
COMMAND	m300	m937	m242	m974	m375	m707	m118	m590	m926	m553	m600	m151	m000	m272	m068	m464
EditPaste	2318	373	50	376	296	315	286	230	274	211	22	55	53	32	22	1
EditClear	1432	393	799	340	233	61	74	2	0	0	265	131	0	6	1	0
EditCopy	1913	277	27	104	101	47	152	145	97	120	26	9	35	20	17	0
EditCut	576	132	8	80	94	284	17	12	100	26	1	25	12	6	3	0
EditUndo	69	76	61	35	18	31	79	24	14	9	3	13	4	14	6	0
EditFind	217	4	1	12	2	4	8	82	1	1	0	0	1	0	2	0
EditSelectAll	147	20	11	4	2	3	8	3	12	5	20	3	0	11	1	0
EditDeleteWor	83	19	1	4	0	0	0	0	0	0	0	0	0	0	0	0
EditReplace	67	1	0	2	0	0	3	3	0	1	0	0	0	0	0	0
EditPasteSpeci	14	1	35	2	0	0	1	0	0	0	0	0	0	0	1	0
EditRedo	4	3	2	5	0	0	3	0	2	0	0	1	0	0	2	0
EditGoTo	11	0	4	2	2	0	0	0	0	0	0	0	0	0	0	0
EditGoToHead	3	2	0	1	0	6	0	0	0	0	0	0	0	0	0	0
EditCopyAsPict	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0
EditAutoText	0	0	0	0	3	1	0	0	0	0	0	0	0	1	0	0
EditDeleteBack	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0
EditBookmark	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0
EditGoBack	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
		Expected														
Count of COMMAND		EMPNMBR														
COMMAND	m300	m937	m242	m974	m375	m707	m118	m590	m926	m553	m600	m151	m000	m272	m068	m464
EditPaste	2328	442	339	329	259	255	214	170	170	127	114	80	36	31	19	0
EditClear	1770	336	258	251	197	194	163	129	129	96	87	61	27	23	14	0
EditCopy	1464	278	213	207	163	161	135	107	107	80	72	51	22	19	12	0
EditCut	652	124	95	92	72	72	60	48	48	35	32	23	10	9	5	0
EditUndo	216	41	31	31	24	24	20	16	16	12	11	7	3	3	2	0
EditFind	159	30	23	22	18	17	15	12	12	9	8	5	2	2	1	0
EditSelectAll	118	22	17	17	13	13	11	9	9	6	6	4	2	2	1	0
EditDeleteWor	51	10	7	7	6	6	5	4	4	3	2	2	1	1	0	0
EditReplace	36	7	5	5	4	4	3	3	3	2	2	1	1	0	0	0
EditPasteSpeci	26	5	4	4	3	3	2	2	2	1	1	1	0	0	0	0
EditRedo	10	2	2	1	1	1	1	1	1	1	1	0	0	0	0	0
EditGoTo	9	2	1	1	1	1	1	1	1	0	0	0	0	0	0	0
EditGoToHead	6	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
EditCopyAsPict	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EditAutoText	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EditDeleteBack	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EditBookmark	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EditGoBack	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Instruction														
Count of COMMAND		EMPNMBR														
COMMAND	m300	m937	m242	m974	m375	m707	m118	m590	m926	m553	m600	m151	m000	m272	m068	m464
EditPaste			More								More					
EditClear	_			_	_	_	_	More	New	New		_	New	_	More	
EditCopy			More									More				
EditCut	_	_	More	_	_	Alt	_	_	_	_	More	_	_	_	_	
EditUndo	_	_	_	_	_	_	Alt	_	_	_	_	_	_	Alt	Alt	
EditFind	_	More	More		More	More	_	Alt	More	More	New	New		New	_	
EditSelectAll	_	_	_	Alt	More	Alt	_	_	_	_	Alt	_	New	Alt	_	
EditDeleteWor			More	_	New	New	New	New	New	New	New	New				
EditReplace	_	More	New	_	New	New	_	_	New	_	New	New				
EditPasteSpeci		Alt	Alt		New	New		New	New	New	New					
EditRedo	_	_	_	Alt	New	New										
EditGoTo		New	Alt	_	_											
EditGoToHead																
EditCopyAsPict	New															
EditAutoText	New															
EditDeleteBack	New															
EditBookmark	New															
EditGoBack																
1.13	0	2	5	0	2	1	0	1	1	1	2	1	0	0	1	0
0.87	0	1	2	2	0	2	1	1	0	0	1	0	0	2	1	0
2.27	4	1	1	0	4	4	1	2	4	3	4	3	2	1	0	0

Figure 4. Computing Tutorial Intervention

A command whose expected value is zero need not be learned, and can be ignored; its indicator is a blank. A command that has an expected value, but is one the individual has never used, is a command the individual probably would find useful if she were to learn it; its indicator is "New." A command whose usage is within normal range of the expected value can also be ignored. In this example, the normal range was determined empirically so as to limit the amount of recommended learning, on average, to two New commands, one More command, and one Alt command. A similar process is used when generating the actual recommendations. The

indicator for a command within normal range of the expected value is an underscore. A command that is used less than expected may be a component of tasks unknown but potentially valuable to the user; its indicator is "More." A command that is used more than expected may indicate ignorance of more powerful ways of accomplishing some tasks; its indicator is "Alt."

Notice that expected values are in some sense, an average, but OWL does not urge the learner towards the skills of the average user, instead, OWL urges the learner towards the pooled knowledge of all users. Even the most knowledgeable individual can learn something from the pooled knowledge of the group. In the example, four New recommendations are made to m300.

These indicators are based on current log data, but as we saw earlier, there may not be enough data in a user's log to determine with confidence that a command is not known to the user. For example, let us take user m590 who has four indicators: two New, one More, and one Alt, for commands EditPasteSpecial, EditDeleteWord, EditClear, and EditFind, respectively. User m590 has logged about 1400 commands. When we calculate the confidence intervals using the method referred to earlier, we find that usage of EditClear is not within the confidence interval around the expected value, but well below it, and that EditFind is also not within the confidence interval, but above, it, so OWL is justified in making the suggestion that EditClear be used more, and in teaching alternatives to EditFind, if there are any. However, the confidence intervals for EditDeleteWord and EditPasteSpecial include the zero value, which means that the lack of use we have observed does not necessarily signal lack of knowledge on the part of our learner; therefore, OWL must present its recommendation that the user learn these commands in a manner that does not insult the user if she should already be familiar with them.

OWL currently recomputes recommendations monthly. Our average user logs about 400 commands per month. Our intention is to present new sets of recommendations as frequently as possible, but only if the new set of tips is significantly different from the previous one, i.e., after collecting sufficient data to observe changes in either the individuals' or the group's use of the software. One way to automate this process might be to recompute each individual's tips daily, but make them available to the user only when they have changed significantly.

To review, OWL compares actual and expected values to develop instructional indicators for each command for each person each month. The expected values are those would appear if the user were behaving exactly as her peers - taken in the aggregate - do. Significant differences between the user's actual and expected values are taken as instructional indicators, i.e., indicators that the user might not know something her peers find sufficiently valuable to utilize with a certain frequency. OWL uses the command name, its frequency of occurrence, and the instructional indicator to make the learning recommendations it presents to the user in the form of Tips. The decision as to which commands to learn, or to learn more about, is left up to the user. If OWL's timing is right, most users will have felt the desire to learn more, but not yet done so, when the Tips are presented to them. The next sub-section describes the Tip Presenter and the Skill-O-Meter.

Providing Feedback to Users

As mentioned, new sets of recommendations, known as *Tips*, are computed monthly. When the new tips are computed, we send each individual an email message telling them that they have a new set of tips. The tip file for each individual resides on the server. Whenever the person decides they want to learn something more about Word, they click on the Tip button on the OWL toolbar, the Tip Viewer (Figure 5) then retrieves the individual's tip file, containing the top ten tips for that person, from the server and displays it to them. The Tip Viewer presents the tip name, a brief description, and a relative ranking of the top ten tips. Users can then click on any tip to learn more about the command. Today, the instruction is elementary: clicking on a tip simply brings up the appropriate Word Help page. We have been field testing the OWL Tip mechanism with about 20 users for several months and are now (October 1999) scaling up the project with the goal of having about 100 users for the remainder of the research.

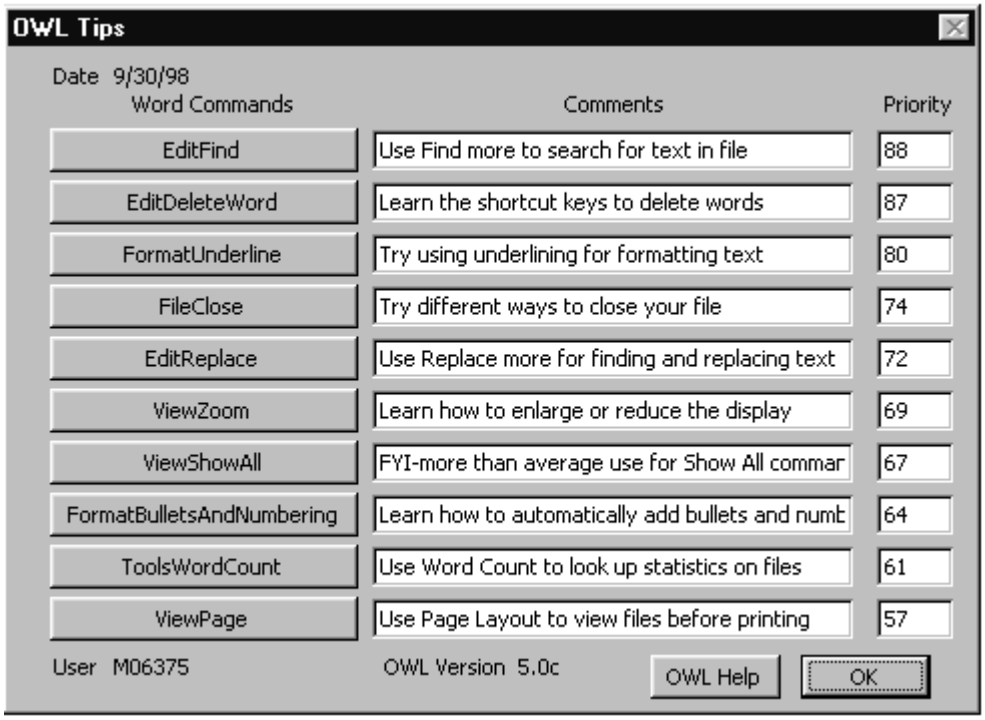

Figure 5. OWL Tip Viewer

OWL Tips focus on what the user should learn next. That is just another way of saying that OWL Tips focus on the user's ignorance, and no one likes to have their ignorance brought to their attention. We have designed another tool intended to present users with a more-complete view of the state of their knowledge, the Skill-O-Meter (Figure 6). Like the skillometers of Anderson (1992) and Lesgold (1992), the Skill-O-Meter will show users what they know, instead of what they don't know. In OWL the Skill-O-Meter will show users what they know compared to the pooled knowledge of a demographic group the users selects themselves, such as users with a particular job title, in a particular department, holding the same degree, etc.

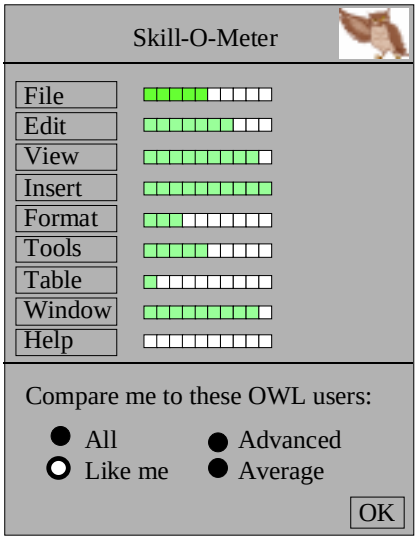

Figure 6. Skill-O-Meter

Discussion

In this section we discuss several issues that have the common thread of comparing expertise or knowledge as it appears in OWL's recommendations with more-familiar concepts of expertise. We start with expertise in OWL

vs task-based expertise, and continue with OWL vs performance support systems, and OWL vs Microsoft's Office Assistant, and close the section with considerations of possible influences on command use frequency.

Expertise in OWL vs Traditional Task-based Expertise

We compare OWL's definition of expertise, which is that expertise consists of knowing a number of commands, to a conventional AI definition, such as Koedinger and Anderson (1993), where expertise consists of knowing one or more plans for attaining a goal, and the plans ultimately decompose to a sequence of actions. For example, the plan to achieve the goal of opening a file might consist of these actions:

menu_select File/Open
list_select <path>
list_select <filetype>
list_select <filename>
button_click "Open"

In this example, knowledge of the same plan is indicated in OWL's view as use of the FileOpen command. It is often the case that text editing plans are indicated by a single command. Furthermore, there is no editing plan for many common writing goals. For example, the general goal of 'revising a paragraph' might be carried out by a vast number of editing goals in any sequence. It is only in instructional situations, for example, when the initial and end states of the paragraph are given, and the student is told to use a specific set of commands to accomplish the revision, that the problem become tractable. Also, all the instruction for the goal of opening a file is located in the Help for the FileOpen command, and OWL need only point the user to this instruction. Finally, as OWL does not provide instruction in moment-to-moment context (as Microsoft's Office Assistant, discussed below, does), it is not necessary to recognize user's plans as they are being carried out.

Nevertheless, we must agree that some goals, such as moving text, are achieved by issuing a sequence of commands, e.g., EditCut followed by EditPaste. Also, if a command is part of a plan for achieving a frequently-appearing goal, and the user does not know the plan, this lack of knowledge may appear as under-usage of that command. Another reason OWL should recognize sequences of commands is to find misconceptions. For example the sequence copy/cut/paste, required in some early editors, can now be replaced by cut/paste. OWL should recognize this misconception and teach the revised task-plan-action sequence. Similarly, some applications open with a new, blank document; it is no longer necessary to close the blank document manually before opening an existing document. For these reasons, we are beginning to explore automated means of detecting and recommending meaningful sequences of commands.

OWL vs Performance Support

A performance support system helps the user do her current job task. It assumes that the task is sufficiently well-understood that (1) the user can define the task and find it in the performance support system, i.e., that user and system have a common name for the task, (2) the organization has devoted resources to creating performance support materials for the task, and (3) that the task has not changed since the time the performance support materials were created. OWL does not provide moment-by-moment performance support but a gradual longer-term increase of skill. OWL is less ambitious than performance support in that it is focused on software tasks rather than job tasks. The organization gets less from OWL than from performance support but it does much less to get it, so the return may be higher. OWL requires little up-front task analysis and is nearly instructional-design free. It is only necessary to ensure that there are pointers into the appropriate places in the Help system for each command that OWL recommends. These can be added as needed.

OWL's advice evolves as users learn about the application. Conventional performance support requires that the knowledge transmitted to the learner already exist. OWL transmits knowledge from user to user as that knowledge - of which commands are the most useful - is acquired by the organization. As soon as anyone finds some function of the software useful, OWL captures that knowledge. The pooled knowledge of all the individuals is greater than the knowledge of any one individual, even the most knowledgeable. Whenever an individual is ready to learn something more, - the next most useful function of their software, given the current state of their knowledge - OWL computes it and transmits it to them.

Finally, performance support systems become obsolete when there are changes in the organizational context and individuals begin to perform new tasks and apply new skills. In contrast, OWL evolves in a second way, it

automatically adapts to changes in the organizational context, that is, when its users begin to use the software in different ways, OWL's recommendations will also change.

OWL vs Microsoft's Office Assistant

A scene that typically comes to mind when discussing automated instruction for software users goes something like this: A user is inefficiently deleting several words, by deleting them one character at a time. The instructional system observes this inefficient action and suggests the individual use the command for deleting a word at a time. Constructing a system capable of carrying out this scenario requires (1) finding and building the useful logical equivalencies in advance, and (2) observing the user and inferring her plan in real time. The current state of the art of this approach is illustrated by Microsoft's Office Assistant. In contrast, OWL simply observes that the user never deletes a word at a time while her peers often do, and suggests that, if she were to learn the command she would find it useful. The knowledge base of the Office Assistant required many staff-years of work to develop (Horvitz, 1997), while OWL acquires much of its knowledge base automatically. The Office Assistant's knowledge base is general and static, while OWL's is organization-specific and evolves as the knowledge of its users evolves. If a user creates an outline by hand, the Office Assistant will suggest using View/Outline instead only if it can determine the logical equivalence of the two activities. In contrast, OWL will suggest the user try View/Outline, if her peers have found it useful. Office Assistant provides instruction in the context of the user's 'doing' while OWL provides instruction in the context of the user's learning.

Possible Influences on Command Use Frequency

There may appear to be several valid reasons why some people use commands that others do not, or use them with a frequency that others do not. One reason for these differences is differences in the tasks they are performing; examples of task-related commands include MailMerge and InsertTable. Another reason is different individual histories with other software with divergent interaction models; Word users with Emacs experience may habitually Copy, Cut & Paste instead of the simpler Cut & Paste. A third reason is stylistic differences or individual preferences; some users will pay attention to formatting their text, while others accept the default settings.

Nevertheless, it is our observation that all users are focused on getting their jobs done. If they have only one method of accomplishing a task they will use it. If they have two or more methods, they will usually choose the more appropriate one, where 'more appropriate' depends on the immediate circumstances. For example, a person familiar only with tabs will create a table using tabs, while a person who knows only the Table command will use it instead, but a person who knows both tab and Table will create a table by choosing the more effective method, depending on the circumstances. History and preferences can help explain a user's choice of commands, but our observations reveal that users who know two ways of accomplishing a task use them both; we believe they perform an unconscious cost/benefit analysis and generally choose the more appropriate method, given the circumstances. Thus we believe that by recommending commands their peers have already found useful, we empower individuals to make more effective choices in their selection of editing methods, regardless of task, history, and preference.

Concluding Remarks

We have presented the OWL recommender system as a means of helping a community of practice pool and share its expertise, where the community is a set of people who desire to become skilled users of their application software. OWL makes individualized learning recommendations, based on the comparison of individual user skills to the pooled knowledge of their peers.

Expertise evolves because users continuously figure out better ways to accomplish their tasks, respond to changes in the software, and to changes in the work environment as well. OWL provides continuous performance improvement to those who do complex job tasks with application software. The system's expert models serve as the organization's memory of expertise.

Analysis of user log data revealed that command usage frequencies for the Word text editor follow a power law. As a result of this power law of usage, careful work is required to distinguish apparent user ignorance or learning

from effects due solely to the amount of information logged. A method was proposed for taking these effects into account.

We further discovered that users tend to acquire commands in sequence. By combining the power law of usage with the acquisition sequence, we can determine the expected usage of each command by each individual: an individualized expert model; and take large deviations from this model as indicators of ineffective software use -- and learning opportunities. Finally, we described two methods, one planned and one implemented, for communicating these learning opportunities to OWL users.

References

Anderson, J., Corbett, A., Fincham, J., Hoffman, D., & Pelletire, R. (1992). General Principles for an Intelligent Tutoring Architecture. In J. Regian and V. Shute (Eds.) *Cognitive Approaches to Automated Instruction*, Mawah NJ: Erlbaum, 81-106.

Cheikes, B., Geier, M., Hyland, R., Linton, F., Rodi, L., & Schaefer, H. (1998) Embedded Training for Complex Information Systems. In B. P. Goettl, H. M. Halff, C. L. Redfield, & V. J. Shute (Eds.) *Intelligent Tutoring Systems 4th International Conference Proceedings*, Heidelberg: Springer-Verlag, 36-45.

Eales, J. & Welch, J. (1995). Design for collaborative learnability. In J. Schnase & E. Cunnius (Eds.) *Proceedings of CSCL '95: The First International Conference on Computer Support for Collaborative Learning*, Mawah NJ: Erlbaum, 99-106.

Gladwell, M. (1999). The Science of the Sleeper: How the Information Age could blow away the blockbuster. *The New Yorker*, October 4, 1999, 48 – 55.

Horvitz, E. (1997). Invited Talk. *Presented at the 1997 User Modeling Conference*, June 2-5, Sardinia, Italy.

Howell. D. (1982). *Statistical methods for psychology*, Boston: Duxbury Press.

Kay, J. & Thomas, R. (1995). Studying long-term system use; computers, end-user training and learning. *Communications of the ACM*, 38 (7), 61-69.

Koedinger, K. & Anderson, J. (1993). Reifying implicit planning in geometry: Guidelines for model-based intelligent tutoring system design. In S. LaJoie & S. Derry (Eds.) *Computers as Cognitive Tools*, Hillsdale NJ: Erlbaum, 15-46.

Landauer, T. (1996).*The Trouble With Computers: Usefulness, Usability, and Productivity*, Cambridge MA: MIT Press.

Lave, J. & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*, Cambridge: Cambridge University Press.

Lesgold, A., Eggan, G., Katz. S. & Rao, G. (1992). Possibilities for Assessment Using Computer-Based Apprenticeship Environments. In J. Regian and V. Shute (Eds.) *Cognitive Approaches to Automated Instruction*, Mawah NJ: Erlbaum, 49-80.

Linton, F. (1990). A coach for application software. *CBT Directions*, March, 22-29.

Linton, F. (1996). Promoting the Organization-Wide Learning of Application Software. *ACM SIGOIS Bulletin*, 17 (3), 70-72.

Linton, F., Joy, D. & Charron, A. (1999). OWL: A Recommender System for Organization-Wide Learning. *MITRE Technical Report MTR 98B0000025V00S00R00*, USA: MITRE.

McCalla, G., Greer, J., Kumar, V., Meagher, P., Collins, J., Tkatch, R. & Parkinson, B. (1997). A peer help system for workplace training. In B. du Boulay and R. Mizoguchi (Eds.) *Artificial Intelligence in Education: Knowledge and Media in Learning Systems, Proceedings of AI-ED 97*, Amsterdam: IOS Press, 183 - 190.

Patton, M. (1990). *Qualitative evaluation and research methods*, 2nd ed., London: Sage.

Resnick, P. & Varian, H. (1997). Introduction to Special Section on Recommender Systems. *Communications of the ACM*, 40 (3), 56-58.

Senge, P. (1990). *The fifth discipline: The art & practice of the learning organization*, New York: Currency Doubleday.

Thomas, R. (1996). Long term exploration and use of a text editor. *Unpublished doctoral dissertation*, Nedlands, Australia: University of Western Australia.

Thomas, R. (1998). *Long Term Human-Computer Interaction : An Exploratory Perspective*, Springer Verlag: New York.

Triola, M. (1983). *Elementary Statistics*, 2nd ed., Menlo Park CA: Benjamin/Cummings.