Continuous Learning Through Inline Training

Brian Krisler
Information Knowledge and Technologies
Raytheon BBN Technologies
Cambridge, MA
bkrisler@bbn.com

Richard Alterman
Department of Computer Science
Brandeis University
Waltham, MA
alterman@brandeis.edu

***Abstract*—Computer workers are in constant tension between meeting their deadlines and learning the tools they use to perform their jobs. Most often, the press to get work down overrides the importance of how to continually use the tool, and thereby improving performance over the long run. The lack of knowledge, however, results in constant interruptions to the workflow as the engineer tries to "bend" the tool to meet their tasks. This paper presents a method to convert these workflow interruptions into targeted opportunities for training that can be leveraged through the use of an inline trainer. An inline trainer is a tool that works in parallel with the main application to promote continuous learning through the introduction of small amounts of information that over time lead to a richer understanding of an application. Data is presented demonstrating that when provided with an inline trainer, users will leverage the tool to improve their daily work, making more productive use of already occurring interruptions.**

Keywords— Adaptive computer learning; Conceptual maps; Inquiry based learning; Lifelong learning

I. INTRODUCTION

From routine email management tasks to complex drafting activities, software plays a critical role in an engineer's daily workflow. To maintain and increase proficiency with these tools, engineers' must continuously learn and adapt as each new task increases in difficulty, requiring more intricate knowledge of the software tools required to perform their job. Training courses help to introduce new and useful functions, however these courses are often detached from real work tasks, resulting in very little absorption of new knowledge. User manuals provide detailed information about applications and their functionality, but their usage requires postponement of actual work to read and digest. As a result, engineers often fail to master the tools they use on a daily basis [1].

A software interface is designed with several usability goals in mind [2]: effectiveness, efficiency, safety, utility, learnability, and memorability. The attainment of these goals depends on the end-user's understanding of the application. As software interfaces continue to grow in complexity with each new version, the need to continuously learn the software becomes essential. However, most approaches to training fail to address the needs and requirements of continuous learning and training.

Software training can be divided into two stages: a) the initial stage, where the training is focused on the development of a basic set of operators for a given application, and b) the advanced stage, where the focus of the training turns toward using the application to perform highly specific tasks.

During the initial stage, a working knowledge of the interface is developed. Then, the user's focus turns to completing real tasks, and the press of "getting work done" prevents her user from progressing beyond this established working knowledge [3], and thus creates a plateau in her knowledge of the application. The failure to develop application proficiency results from a strong dependence on basic operators that is typical for a user always under the pressure of a deadline. From here, users rarely realize the full functional power [4] of the tool.

When an engineer relies on a limited functional knowledge of a software application, interruptions in her main task become a regular occurrence. For a menu-driven system these slow downs manifest as searching for known or "hoped for" operations. These interruptions can be reduced by increasing her knowledge of the software; increasing the users' knowledge of the underlying conceptual model of the application increases their power of thinking [5], which in turn improves their productivity with the application. For example, learning to invoke a frequently used operator via a keyboard shortcut instead of performing a menu selection improves efficiency while at the same time reduces interruption costs.

Our research [6] investigates methods for increasing a user's application skill level by introducing small bits of training into the flow of work. This paper presents an *inline trainer*, LEAFs that works in tandem with a user to incrementally increase application knowledge on a daily basis as a part of routine work. The goal of the LEAFs trainer is to address the many shortcomings of existing training methods while leveraging the possible opportunities inherent in workflow interruptions.

An inline trainer [7] introduces new application knowledge into the naturally occurring interruptions that are part of a continuous sequence of operations. The addition of an inline trainer into the workflow enables a user to regularly increase their knowledge about software tools they use without distrubting the work flow. An inline trainer enables the user to leverage the interruptions that are an intrinsic part of the interaction. Overtime these small pieces of knowledge will combine to form larger concepts, eventually leading to an increase in performance.

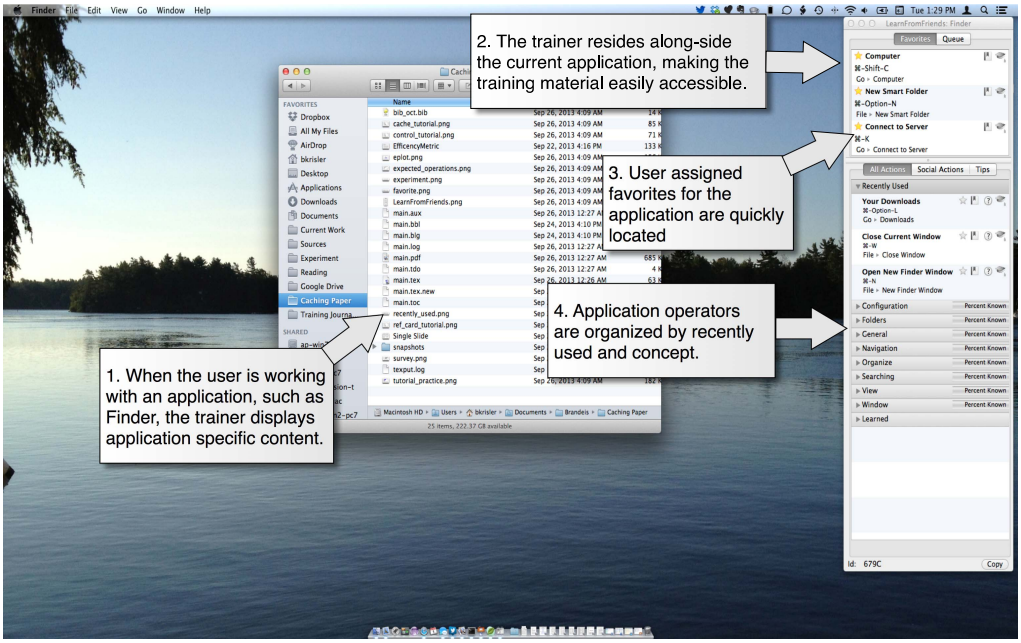

Fig. 1. The LEAFs inline trainer is a training tool that works in parallel to the main application, providing users with the ability to locate, use and learn more about the operators in the applications they use on a daily basis.

This paper presents data collected from a two-day experiment that compares task performance between subjects with and without access to an inline trainer. The data shows that for all subjects, interruptions in the work activity regularly manifest during typical menu interactions. As discussed in detail below, in a normal workday situation, the average user does not extend these workflow interruptions to learn; these results are replicated in the experiment for subjects without access to the inline trainer.

At issue is whether subjects with access to an inline trainer would establish an alternate pattern of behavior. Would the inline trainer reduce the "extra work" typically required by training for the subjects to decide to expand their knowledge of an application as they "worked" on the experimental tasks? The data shows that subjects with access to the inline trainer, did in fact use it. We will analyze the data to show what kinds of training aids the subjects were willing to use and how these aids impacted their learning.

II. BACKGROUND

To truly master an application, interface training is essential. When a user interacts with a software application, she often has one or more goals in mind. She is not interacting with the computer for the pure joy of learning [8]. Because of this necessity to complete real work, the quickest approach to goal completion is often taken. While effective in the short-term, this approach leads to problems in the long-term due to a phenomena known as the Active User Paradox [3].

The Active User Paradox is the result of two competing biases: the production bias and the assimilation bias. A *production bias* occurs when a user approaches an activity with the knowledge that a more efficient means to attaining the goal probably exists but the effort required to locate and digest the appropriate training material is often not worth the perceived reward. Instead of expending some effort to locate a more efficient approach to solving the problem, she instead attempts to solve the problem using her existing knowledge about the application. An *assimilation bias* is when a user relies on existing knowledge to try to solve a new and similar problem. Since the existing knowledge help her to produce a successful outcome in the past, it is likely this approach will prove successful again.

The active user paradox presents a performance issue. When the user assimilates to using sub-optimal procedures, she reaches a plateau in her performance [9]. By not seeking out a more efficient means to solving a problem, she rarely realizes the full functional power of the software [4]. In the short-term, she can complete the tasks necessary to satisfy her goals; however, the long-term effects are detrimental to efficiency and productivity due to her limited understanding of how the application works.

When a user interacts with a software application she has a mental model of what the software does and how it works. It is this model that drives her interaction with the system. Competing with the user's mental model, is the conceptual model [10]. The conceptual model is the designer's view of the software. Between the user's mental model, and the designers' conceptual model is a gap. This gap consists of the gulf of execution and the gulf of evaluation. The larger these gulfs, the more issues the user encounters as she attempts to use the application for daily activities.

These issues manifest themselves as workflow interruptions.

Interruptions are common. They are so common that it has been observed that workers switch tasks due to an interruption about every 12 minutes [11]. Interruptions vary [12] in size,

and are often detrimental [13], with the worker easily losing their train of thought, and over 40% of interrupted tasks [14] never getting resumed. However, interruptions are not always a hindrance [15] and can actually aid task completion.

An inquiry is a type of beneficial interruption. Inquiries occur when the user does not have enough knowledge about an application to efficiently proceed with the task [16], referred to inquiries as *knowledge errors*. Here the interruption is the result of unnecessary sub-actions due to insufficient knowledge; for example, not knowing the exact location of an operator on the menus introduces unnecessary searching. When an insufficient knowledge interruption occurs, the user must redirect her attention from the main activity to the interruption.

In [17], Bødker identified two types of redirection: *breakdowns* and *focus-shifts*. Breakdowns are interruptions that result from the tool not performing as expected, causing the user to redirect her attention from the main task to the tool. Until the source of the breakdown is resolved, the user remains focused on the cause of the breakdown, unable to proceed. Focus shifts, on the other hand, are more deliberate changes in activity and occur at points of articulation, such as the searching that occurs when looking for a new operator on the menu.

A breakdown is a harsh interruption that is the result of an unexpected event. The occurrence of a breakdown often places the user in a distracted state, where her resources are redirected towards solving the breakdown. Because of this distracted state, breakdowns are not an appropriate time to introduce new knowledge to the user. Focus-shifts on the other hand are inquiry interruptions, triggered at an articulation point, which make them opportune moments to introduce new application knowledge.

III. LEAFS INLINE TRAINER

The LEAFs inline trainer is a custom developed, native OS X application made available to a user that allows her to leverage focus-shifts as learning opportunities. When a user interrupts their main task to focus on how to do something with an application, instead of perform and forget, they learn and perform. The LEAFs trainer was designed to run in parallel to the main application. (i.e., Finder, Safari, and Mail), providing a user with the ability to quickly navigate between task-work and training (Fig. 1).

LEAFs presents the user with six methods for obtaining quick and succinct training.

A. Favorites Cache

The top of the LEAFs user interface contains a *Favorites Cache* (A in Fig. 2), a learning aid designed to provide the user with an area to store and recall, with a quick glance, features and operators they are most interested in learning. The favorites cache contains all of the information necessary to quickly identify and perform the operation.

B. Recently Used Cache

Below the favorites cache is the *Recently Used Cache*. This learning aid (B in Fig. 2) offered a supportive path to re-finding

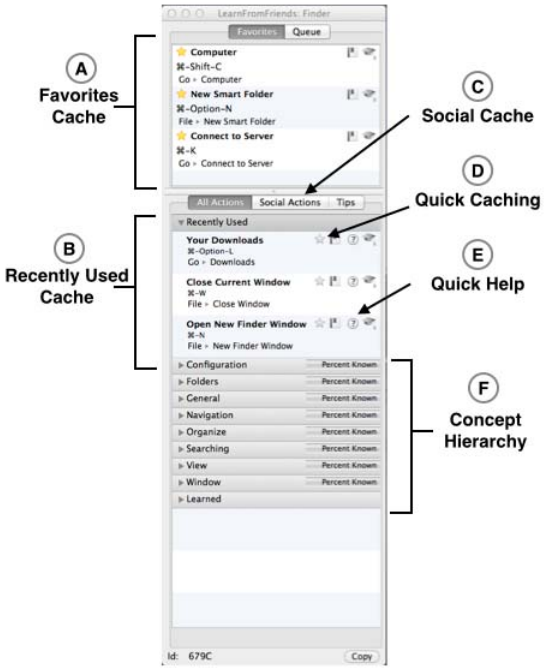

Fig. 2. The LEAFs inline trainer consisted of 6 different learning aids designed to promote and faciliate inline training.

previously used items. Each entry in the cache includes the name of the previously used operator, its keyboard shortcut, menu path, and fast "tooltip-like" access to the application help that did not require launching the application help system.

C. Social Cache

The *Social Cache* (C in Fig. 2) offered another perspective to an application's operators. This learning aid displayed operators ranked on their popularity among a peer-based social group, providing the user insight into how others were finding value in the application. The social cache promotes inline training by providing the user with an "over-the-shoulder" view into how their peers interact with the same application(s).

D. Quick Caching

The *Quick Cache* learning aid (D in Fig. 2) provides the user with an easy to use method for adding an operator to the favorites cache. The quick cache button resides next to every operator throughout the LEAFs trainer, making it straightforward for users to quickly pin and learn an operator without having to search and re-search the user interface.

E. Quick Help

The *Quick Help* (E in Fig. 2) learning aid provided the user with quick access to operator specific application help. Quick help promoted inquery learning by allowing the user to quick view within a small pop-up window more descriptive information about an operator without having to fully task switch into the larger application help system.

F. Concept Hierarchy

The *Concept Hierarchy* (F in Fig. 2) facilitates learning by presenting a conceptual grouping of the application's features. It was designed around application specific actions like: finding, searching and organizing. The concept hierarchy differs from the existing menu structure in that the operators are organized by function or concept, unlike the existing menu structures, where a more familiar and cross-application uniformity is maintained. For example, the Find operator, located within the "Searching" concept is typically found in an application's "Edit" menu – a completely different concept. The concept hierarchy promotes inline training by allowing the user to visualize the conceptual grouping of operators, allowing the user to develop a deeper understanding of the application.

LEAFs provided the user with a collection of individual, easy-to-use learning aids that worked in conjunction with each other to provide a unified, continuous learning experience.

As the user changed between applications (i.e., switched from Finder to Safari) the content of the trainer changed correspondingly. This ensured that the displayed information was relevant to the current application.

IV. EXPERIMENT

The experiment design consisted of three task sets staged to replicate typical application interactions such as working with the file system, using a web browser, and interacting with an email client.

Depending upon the skill level of the subject, one or more operators were required to complete each task. For example, one task required subjects to locate information on the Internet and then send an email containing a link to the page. Solving this task could be accomplished using copy and paste, or using the more optimal mail link operator in the web browser.

For the experiment, three test conditions were established: 1) a control group which would be left to its own devices; 2) a reference card group offered a standard, paper-based summary of features; and 3) an inline trainer group given access to the LEAFs trainer.

All subjects, regardless of condition completed the first task set without any additional training. This established a baseline comprehension against which we were able to analyze. Prior to the second task set, all subjects received a training manual, which summarized some of the more advanced features of the essential OS X applications. By providing this traditional self-help, all subjects had straightforward access to the optimal solution for all the experiment's tasks.

The experiment was conducted over the course of two consecutive days. The objective here was two-fold: could subjects remember newly discovered features on the follow-up day, and did LEAFs users continue to use the trainer on day two?

A. Subjects

Subject recruitment was carried out through flyers and emails. A total of 21 male and female students participated, with a mean number of college years attained as 3.22 (SD=1.22). Subjects completed a survey assessing their own skill level and daily computer usage. A majority classified themselves as having vast computer experience, and that they used a computer "several times a day", however results from the short computer anxiety scale [18], demonstrated the total subject-base as having some level of computer anxiety; a mean of 3.87 (SD=1.94) out of 6.

On the first day of the experiment, each subject was briefed on the procedure and randomly assigned to a condition. Each group containing seven subjects, however two of the subjects in the inline trainer group did not return for the follow up day. All of the subjects were compensated $15 for their participation.

All participants had unfettered access to help, either through manuals, online help, or the Internet. In addition to these resources, subjects in the reference card group were given a typical software reference card of keyboard shortcuts for the three primary OS X applications. Subjects in the inline trainer group were presented with a 10-minute screencast tutorial demonstrating the basic features and functionality of the training tool.

B. Tasks

Twenty-five tasks were asked of the research participants. These fell within the three main categories of working with files, using the web browser, and interacting with email. These were also tasks that could be performed through a mixture of standard or lesser- known interface operations. The expectation was that all of the subjects would have some inefficiencies in how they completed the work.

The experiment placed minimal constraints on the subjects' workflows. Each set of tasks was designed to capture how subjects worked on these tasks outside of typical experiment's boundaries. During the experiment, subjects were free to complete the tasks as they wish, in the order they preferred. The re-introduction of operators created a repetitive pattern with the intent of leveraging new knowledge. Subjects were also instructed to order the tasks as they saw fit, tackling easier ones first if need be. And, subjects were told they could abandon a task if it was too hard.

C. Collected Data

To evaluate how each subject interacted with the system during the experiment, an event logger was developed that recorded important experiment interactions. Each observed event contained nine fields to facilitate offline analysis (Fig. 3a).

Fig. 3b is a snippet from a collected transcript. In this block, a sequence of actions performed by the subject 387C during task 1.1 are presented. In line 1, the event recording tool was launched by the experiment proctor. Five minutes later, the application Finder became the primary application, and the File menu was opened (line 2). In the File menu, the New Folder operator was selected with the mouse (line 5).

By analyzing the transcripts in this form, we were able to recreate the steps taken by each subject and discover how they searched, found, and executed the required operators for completing each task. From this analysis, we were able to assess the utility of an inline trainer in conjunction with everyday tasks. To facilitate the data analysis stage, a series of scripts were developed that allowed for quick searching and processing of the event logs.

RESULTS

The data shows that even during the completion of simple experiment tasks, subjects became preoccupied with the task at hand, relying on aimless searching of the menu instead of leveraging any of the available help systems to solve unknown problems. These traditional learning aids, such as user manuals, application cheatsheets were readily available, yet they proved infrequently used underscoring their ineffectiveness as everyday learning aids.

The data also shows that the experiment group with access to the LEAFs inline trainer did alter their behavior, and leveraged the lack of knowledge as an opportunity to learn more about the application. Instead of aimlessly searching an applications menu system for a solution, the inline trainer subjects used the tool to their advantange, locating unknown operators, and staging these operators for easier recall and continued learning.

A. Searching the menu and lacking knowledge

The experiment presented subjects with tasks that required modifications to their existing routines. These modifications often necessitated that the subject locate a new, possibly unknown, operator. It was observed throughout the data that when a subject did not have the knowledge necessary to quickly locate the required operator, they referred to the menus for a solution. In one instance, where a subject was trying to discover a way to convert all of an email's text to uppercase (see Fig. 4), the menu was the primary method of discovery. During the search for the Make Upper Case operator, the participant can be seen opening and re-opening the same menu (Fig 4., lines 5 and 8). When a plausible operator was found, trial-and-error was used (Fig. 4., line 6 and line 12) to evaluate the outcome.

		Observed User Action	
	(sec)	Menu Search	Operator Performed
1.	0	*Edit*	
2.	5	*Attachments*	
3.	6	*Find*	
4.	7	*Spelling and Grammar*	
5.	8	*Substitutions*	
6.	12		*Text Replacement*
7.	17	*Edit*	
8.	19	*Substitutions*	
9.	19	*Transformations*	
10.	20	*Speech*	
11.	20	*Transformations*	
12.	33		*Make Upper Case*

Fig. 4. Subject's often relied on trial-and-error to locate an operator. Here, a subject is observed searching for an operator to make all text upper case. Lines 1-6 highlight the subject searching the menu and then trying the Text Replacement operator without success. The subject then returns to searching and eventually locates the Make Upper Case operator.

Operator	Expected	Observed (mean)	Difference
Google Search	13	1.80	11.2
New Message	5	2.44	2.56
Save As	3	2.00	1.00
Mail Contents of This Page	5	4.05	0.95
Mail Link to This Page	4	1.58	0.94
Compress	6	5.29	0.71
Center	2	2.00	0.00
Insert Bullet List	1	1.00	0.00
Downloads	1	1.00	0.00
Make Upper Case	1	1.00	0,00
Forward as Attachment	1	1.00	0.00
Open Location	3	0.00	0.00
Find	2	0.00	0.00
New Folder with Selection	4	0.00	0.00
Paste as Quotation	2	2.17	-0.17
Reply	9	9.19	-0.19
Forward	1	1.47	-0.47
Documents	1	1.86	-0.86
Get All New Mail	1	2.00	-1.0
New Tab	2	3.10	-1.1
Send	20	21.29	-1.29
Close Tab	1	3.75	-2.75
Create New Folder	1	4.28	-3.28
Copy	5	16.38	-11.38
Paste	3	16.21	-13.21

Fig. 5. An analysis of the most optimal method for performing each experiment task was performed and compared to the mean subject performance. This analysis demonstrated the subjects high reliance on an applications menus as a method of knowledge discovery.

This menu dependence resulted in extensive and ineffective searching, the finding and forgetting of operator locations, and in some instances, task failure. To evaluate menu dependency, a search cost metric was developed. Search cost measured the amount of extra work expended by a subject to perform an operator by computing the difference between the observed interface actions and the optimal interface actions.

A search cost analysis over the entire data set showed that

Label	Description	Example
TID	The task id	1.1
SUBID	The subject id	708A
DAY	The day of the observed event.	1
TIMESTAMP	The timestamp for the event	21113100500
APPLICATION	The active application for the event	Finder
ACTION	The observed action	File/New Finder Window
INTERFACE	The event interface (Keyboard (0), Mouse (1), Unknown (2))	1
DETAILS	Auxiliary information about the event	Marked Favorite
GROUP	The experiment group of the subject	Control

(a) All recorded events consisted of nine fields to facilitate data analysis.

#	TID	SubId	Day	Timestamp	Application	Action	Interface	Details	Group
1	SETUP	387C	1	10/25/2012 14:05:38	LFF		2	LAUNCHED.001	Control
2	1.1	387C	1	10/25/2012 14:10:28	Finder	MENU_OPENED	2	File	Control
3	1.1	387C	1	10/25/2012 14:10:28	Finder	MENU_CLOSED	2		Control
4	1.1	387C	1	10/25/2012 14:10:31	Finder	File/New Folder	1		Control

(b) A segment from a recorded experiment transcript.

Fig. 3. Transcript data was collected throughout the experiment, and then coded for post-analysis. (a) shows the nine unique fields used to encode each interaction the subject performed. (b) is an encoded transcript segment.

extra work was often required to perform each operator (M=3.68, SD=4.93). Further analysis revealed that this extra work translated into time inefficiencies. The amount of time spent searching the menus compared to the total experiment time revealed an average of 9.17% of the experiment time was spent searching the menus.

A strong menu reliance led to an overuse of known operators. Basic operators such as copy and paste were used in multiple situations to compensate for a user's lack of knowledge of existing more powerful operators that could significantly reduce interface work. An over-reliance on these basic procedures deterred discovering new and more efficient methods for completing tasks.

To measure the extent of basic operator over-reliance, a task operator analysis was performed. Fig. 5 lists all of the expected operators during an optimal performance of the experiment. The expected column lists the total expected observations over the course of the experiment. For example if performed optimally, the Mail Contents of This Page operator would have been used five times. The observed column presents the average actual observed count of each operator. For Mail Contents of This Page, 4.05 demonstrated an underutilization. Difference presents the expected minus the observed, highlighting operator over and underutilization.

Fig. 5 demonstrates the extent of the over-reliance on general procedures such as copy and paste. If each task were performed optimally, these operators would have only been required five and three times respectively, however their actual usage exceeded that, with an average usage being 16.38 and 16.21.

After extensive but fruitless searches, several research participants quit their tasks entirely. For example, one experiment task asked the subjects to move files from the desktop into a newly created folder, compress the folder and email it to a specific destination. Many subjects struggled because they could not find the Compress operator. One subject was observed performing a vast amount of searching, opening 68 menus over the course of three minutes trying to find Compress. Located in the File menu, Compress is clustered [19] correctly in the menus, yet locating the operator still proved to be a daunting task. In fact, this subject opened and reviewed the File menu 14 times over the course of the search, and at one point even spent 10 seconds reviewing its contents.

When the search model results in an unsuccessful outcome, the interruption turns into a complete task breakdown. This breakdown then has larger implications on the individual's workflow. During the experiment, failed searching was prevalent, with a total of 59 failed searches observed, accounting for 10% of the mean time to complete the experiment.

B. Traditional sources of application knowledge

During the experiment other resources for learning about the application were available, but the subjects rarely leveraged this resources.

All of the subjects in the reference card group were given a printed reference card. This one-page sheet contained all of the functionality, and associated keyboard shortcuts necessary to complete the experiment. However throughout the experiment, the proctor observed the participants and noted that none of the subjects referred to the reference card. For this group, the answers were in plain view, yet no one considered the printed sheet as a viable option for solving the experiment tasks.

Subjects were also free to use any other existing resource, such as the built-in application help, and the Internet to solve a task. An analysis of the transcripts revealed that no subjects in any of the conditions referred to the built-in application help.

There were 10 observed instances of subjects attempting to use an Internet search to discover how to perform a task, with mixed results. Overall, the success rate for this technique was low, with only 30% of the searches resulting in a solution to the problem. This low success rate was often the result of the subjects inability to construct an appropriate search term to describe the problem [20].

C. The LEAFs inline trainer altered behavior

The LEAFs inline trainer provided subjects with six different types of learning aids. Each aid expanded the opportunities for an individual to explore and learn more about the applications they currently use on a daily basis. The data showed that when given access to an inline trainer, subjects did alter their behavior to more efficiently learn about, and improve their usage of everyday applications.

1) Favorites Cache

The LEAFs trainer allowed subjects to quickly and easily cache potentially interesting and useful operations for future recall. Fig. 6 presents transcript segments of subjects caching operations for future recall. In Fig. 6a, a subject is searching for an operator (1, 2, and 3). Once located, the operator is performed (4), and then immediately cached (5) for future recall. Later in the experiment, the subject leveraged the inline trainer to perform the operator without any searching (6 and 7).

The favorites cache was a frequently used feature by subjects. The favorites cache was used to reduce the relocation of a previously found operator. In most instances, subjects would locate and perform an operation. Once performed, they would use the favorites feature to cache the operator for future use.

	Time	Actions Performed by subject	
	(sec)	***LFF*** Actions	**Finder** Actions
1.	0	Expand *Configuration*	
2.	2	Collapse *Configuration*	
3.	4	Expand *Organize*	
4.	8		Perform *Arrange By Kind*
5.	31	Cache *Arrange By Kind*	
		...	
6.	226		Perform *Arrange By Kind*
7.	227		Perform *Arrange By Kind*

(a) Discovering and caching *Arrange By Kind* operator

	Time	Actions Performed by subject	
	(sec)	***LFF*** Actions	**Mail** and **Finder** Actions
1.	0		Perform *Send*
		start next task	
2.	3	Expand *Recently Used*	
3.	9	Cache *Send*	
4.	12	Cache *Get All New Mail*	
5.	13	Cache *Reply*	
6.	19	Cache *Paste as Quotation*	
7.	25	Cache *Forward*	
8.	50	Expand *Organize*	
9.	53		Perform *Documents*

(b) Caching multiple *recently used* operators

	Time	Actions Performed by subject	
	(sec)	***LFF*** Actions	**Finder** Actions
1.	0		Perform *Mail Link to This Page*
2.	7		Perform *Show Reading List*
3.	11		Perform *Show Reading List*
4.	15	Expand *Recently Used*	
5.	22	Cache *Mail Link to This Page*	
6.	24	Cache *Mail Contents of This Page*	
7.	25		Perform *Mail Contents of This Page*
8.	31		Perform *Send*

(c) Discovering and caching multiple operators

	Time	Actions Performed by subject	
	(sec)	***LFF*** Actions	**Finder** Actions
1.	0		Perform *Reply*
2.	3		Perform *Paste*
3.	5		Perform *Select All*
4.	30	View the *Social Cache*	
5.	39	Cache *Make Upper Case*	
7.	55		Perform *Make Upper Case*
8.	57		Perform *Send*

(d) Discovering and caching the *Make Upper Case* operator

Fig. 6. Caching is an important learning aid used throughout an inline trainer. The transcripts demonstrated multiple instances of subjects leveraging caching.

2) Recently Used Cache

Like the favorites cache, the recently used cache was also a frequently used learning aid throughout the experiment. This cache was used to both relocate operators for invocation, as well as to move just performed operators into the favorites cache for future quick recall (Fig. 6c). That data demonstrated that when given access to an easy recall to previously used operators, subjects took the extra steps required to reduce future effort in relocating an operator.

3) Social Cache

LEAFs also incorporated a social cache that allowed subjects to see how others used the inline trainer. For this experiment, the social cache information was derived via two sources. The cache was seeded with a few potentially useful operators prior to the experiment, and since the same laptop was used throughout the experiment, the cache collected performed actions throughout the experiment, with at least one subject discovering, and leveraging this feature during the experiment. Fig. 7 shows a transcript of a subject using the social cache to find the *Move to Trash* operator. In this transcript, the subject starts by reviewing the *Organize* concept (line 1), after spending 23 seconds with no luck finding an appropiate operator, the social cache is referenced (line 2)

	Time	Actions Performed by subject	
	(sec)	***LFF*** Actions	**Finder** Actions
1.	0	Expand *Organize*	
2.	23	View the *Social Cache*	
3.	43	Cache *Move to Trash*	
4.	57		Perform *Move to Trash*
5.	61		Perform *Move to Trash*
6.	64		Perform *Move to Trash*
7.	66		Perform *Move to Trash*
8.	68		Perform *Move to Trash*
9.	70		Perform *Move to Trash*
10.	90	Cache *Empty Trash*	

Fig. 7. The Social Cache is a learning aid that allows users to see how peers utilize the same software. In this transcript segment, a subject uses the social cache to locate the Move to Trash operator.

where the *Move to Trash* operator is found, cached (line 3) and then performed multiple times (4-9). Fig. 6d presents another example of a subject using the social cache to find an operator. In this instance, the subject used the learning aid to quickly discover an operator that proved to be one of the harder tasks during the experiment, making all of the text in an email upper case. Knowing the potential difficulty of the task, this operator was pre-seeded to the social cache by the experimenters.

4) Quick Caching

Quick caching provided the subjects with the ability to quickly move an operator into their favorites cache. There were multiple unique observances of quick caching used throughout the experiment. The transcript in Fig. 5b, shows a subject using quick caching to stage the trainer for the experiment. Event lines 3-7, show the user adding multiple operators to their favorites cache in a single batch. Later in the experiment, the same subject was observed performing these operations, quickly via the associated keyboard shortcut, demonstrating the utility of caching. In another transcript (Fig. 5c), a subject was observed adding an operator to the cache after it had been used. In this instance, the subject also leveraged the recently used cache demonstrating three inline learning aids used in conjunction: recently used cache to relocate a previously used operator, quick cache to move the operator to Favorites and then Favorites cache to recall the operator.

5) Quick Help

Quick help allowed subjects to quickly review the description of an operator. This feature reduced the overhead to investigating and discovering new operators. Analysis of the transcripts revealed a few instances of subjects leveraging quick help to learn more about unknown operators. These interactions suggest that reducing the amount of in-interruption work required to learn more about an operation will lead to discovery and increased operator usage.

6) Concept Hierarchy

The concept hierarchy presented subjects with a discovery method that allowed them to find operators by concept, such as searching or annotating. Transcript analysis of subjects with the LEAFs tool demonstrated that subjects were more likely to utilize the concept hierarchy to locate an operator, than the menu system. When used, the concept hierarchy led to quicker discovery, and more successful task completion.

V. Discussion and conclusion

All users, irrespective of skill level, struggle with the problem of learning to use an application. As they work, they frequently rely on the menu to discover new knowledge or to re-find operators they previously used. The data collected in this study documents the regular occurrence of these kinds of menu-related interruptions throughout the completion of routine tasks. Three test groups were observed performing the same tasks, with on test group given access to an inline trainer. What was striking was the fact the subjects in the inline trainer condition actually leveraged the inline trainer to complete the experiment tasks. They were under no obligation to use the trainer, and there was no value for them to actually invest any level of effort into learning to better use the tools, yet they still did.

Subjects without access to the inline trainer could have chosen to use other resources – like an Internet search, the help system, or a reference card – but the data demonstrated that this did not regularly occur.

By incorporating an inline trainer into the workflow, some of the problems that emerged for users to better understand an application were offset, leading to better usage. One issue is whether or not an inline trainer will be used. In this study, evidence was presented demonstrating subjects using, and re-using an inline trainer to discover and learn new operational knowledge. Another issue is the amount of extra work the trainer imposes on the users. Some of this extra work can be amortized over an extended period: there is an initial cost to doing the extra work to add something to a cache, but over time the cost of accessing a cached operator is reduced. The key is that the initial extra work required to use the trainer must be small enough to fit into an acceptable window of interruption for the user.

Interruptions are common and expected in the modern workplace. However not all interruptions are created equal, and one type of interruption, incurred from software menu access, creates a short disruption which under the proper circumstances may be sufficient enough to accommodate a little training. Thus, opportunistic leveraging of the interruption could lead to more effective usage of the tool in the long term, with little impact on short-term work. The inline trainer presented here leveraged these menu-produced interruptions to present new application knowledge, allowing the users to expand their conceptual understanding of the tool.

This study has shown that leveraging the small, but frequently occurring interruptions expected in daily software interaction, users could discover new and more efficient methods for solving typical tasks. By providing the user with a means to quickly identify and address the cause of the interruption, a more robust mental model of the software could be developed, allowing the user to focus more on the domain, and less on the tool.

References

[1] S. Gupta, R. P. Bostrom, and M. Huber, "End-user training methods: what we know, need to know," *ACM SIGMIS Database*, vol. 41, no. 4, pp. 9–39, 2010.

[2] Y. Rogers, J. Preece, and H. Sharp, *Interaction Design*. Wiley & Sons 2011, 2007.

[3] J. M. Carroll and M. B. Rosson, *Paradox of the active user*. The MIT Press, 1987.

[4] L. Olfman and M. Mandviwalla, "Conceptual versus procedural software training for graphical user interfaces: a longitudinal field experiment," *Mis Q.*, pp. 405–426, 1994.

[5] M. T. Chi, R. Glaser, and E. Rees, *Expertise in problem solving*. Learning Research and Development Center, University of Pittsburgh Pittsburgh, 1981.

[6] B. Krisler and R. Alterman, "Training towards mastery: overcoming the active user paradox," in *Proceedings of the 5th Nordic conference on Human-computer interaction: building bridges*, 2008, pp. 239–248.

[7] Krisler, Brian, "Continuous Software Training with Three Inline Trainers," PhD thesis, Brandeis University, Massachusetts, 2014.

[8] J. M. Carroll, P. L. Smith-Kerker, J. R. Ford, and S. A. Mazur-Rimetz, "The minimal manual," *Hum.-Comput. Interact.*, vol. 3, no. 2, pp. 123–153, 1987.

[9] J. R. Anderson, M. Matessa, and C. Lebiere, "ACT-R: A theory of higher level cognition and its relation to visual attention," *Hum.-Comput. Interact.*, vol. 12, no. 4, pp. 439–462, 1997.

[10] D. A. Norman, *The design of everyday things*. Basic books, 2002.

[11] V. M. González and G. Mark, "Constant, constant, multi-tasking craziness: managing multiple working spheres," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2004, pp. 113–120.

[12] D. C. McFarlane and K. A. Latorella, "The scope and importance of human interruption in human-computer interaction design," *Hum.-Comput. Interact.*, vol. 17, no. 1, pp. 1–61, 2002.

[13] G. Mark, V. M. Gonzalez, and J. Harris, "No task left behind?: examining the nature of fragmented work," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2005, pp. 321–330.

[14] B. O'Conaill and D. Frohlich, "Timespace in the workplace: Dealing with interruptions," in *Conference companion on Human factors in computing systems*, 1995, pp. 262–263.

[15] J. Jin and L. A. Dabbish, "Self-interruption on the computer: a typology of discretionary task interleaving," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2009, pp. 1799–1808.

[16] D. Zapf, F. C. Brodbeck, M. Frese, H. Peters, and J. Prümper, "Errors in working with office computers: A first validation of a taxonomy for observed errors in a field setting," *Int. J. Hum.-Comput. Interact.*, vol. 4, no. 4, pp. 311–339, 1992.

[17] O. W. Bertelsen and S. Bødker, "Activity theory," *HCI Models Theor. Framew. Multidiscip. Sci.*, pp. 291–324, 2003.

[18] D. Lester, B. Yang, and S. James, "A SHORT COMPUTER ANXIETY SCALE 1," *Percept. Mot. Skills*, vol. 100, no. 3c, pp. 964–968, 2005.

[19] K. L. Norman, "Better design of menu selection systems through cognitive psychology and human factors," *Hum. Factors J. Hum. Factors Ergon. Soc.*, vol. 50, no. 3, pp. 556–559, 2008.

[20] M. Ekstrand, W. Li, T. Grossman, J. Matejka, and G. Fitzmaurice, "Searching for software learning resources using application context," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 195–204.