CLAS: An Approach for Full Use of Application Software

Ken-ichi Matsumoto Shuuji Morisaki Akito Monden Koji Torii
Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, Japan
+81-743-72-5311 +81-743-72-5312 +81-743-72-5312 +81-743-72-5001
matumoto@is.aist-nara.ac.jp shuuji-m@is.aist-nara.ac.jp akito-m@is.aist-nara.ac.jp torii@is.aist-nara.ac.jp

ABSTRACT

Many of the latest application software provide users with a large number of functions; however, the full set of functions is too large for diverse users. Most of users usually use only 10% of the full set of functions provided by application software. This paper describes an approach “Community-based Learning for application software (CLAS)” to let users learn useful functions at low cost for the purpose of improving the users' productivity in using application software. In the CLAS, community members mutually compare their own histories of software function executions, which can be collected automatically, in order to identify a set of candidate functions that should be learned. It is based on the assumption that among the functions that each member of a community uses, some differences in the software usage experiences of each member are observed and the function belonging to the difference is useful for the member to perform his/her current task. The experimental results with a prototype system show both novice and experienced members can more easily obtain usage knowledge of the new functions by the CLAS than a conventional online help system. More than half of functions learned by using the CLAS are new functions whose existences were unknown to the members and cannot be found by the conventional online help system. In the CLAS, users can share the knowledge of software usage in a systematic and effective way.

Keywords

History of software usage, Software usage processes, Experimental evaluation, Application usage monitoring

1 INTRODUCTION

Application software, such as Microsoft Office, Adobe Photoshop, etc., provides users with a very large number of functions. For example, Microsoft Word 2000 has more than 1,000 menu items and short-cut buttons. Although some menu items and/or short-cut buttons may be assigned to one function, we have observed that Microsoft Word 2000 provides users with several hundred functions at least.

Most of users usually use only 10% of the full set of functions provided by application software. In the pilot experiment we performed, a subject used only 70 functions out of several hundreds functions provided by Microsoft Word 2000 in creating a 200-page software design document (the total number of characters was about 100,000 characters, and the total number of charts was 72).

All users do not learn eagerly the new functions even if these functions are useful for them and improve their productivity in using application software [1]. It is because costs to learn new functions of the application software and to support it are very large for both users and developers. Fischer classified the function that user should learn into the following three types [2]:

Type I: Function that the user knows vaguely and uses only occasionally,

Type II: Function that the application software provides and the user believes its existence in application software, but are never actually used,

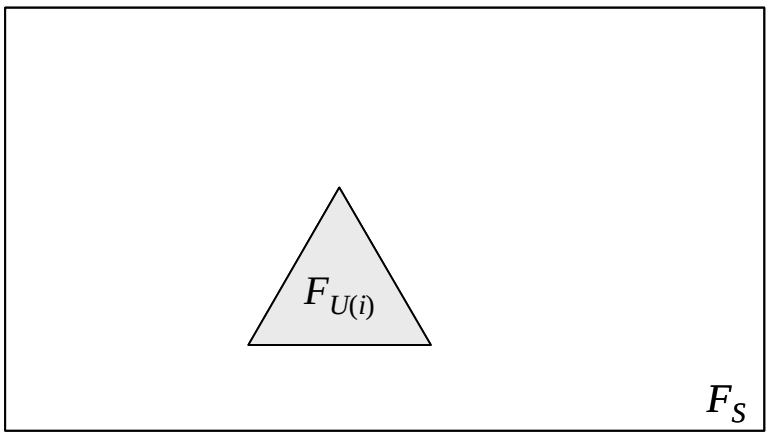


Figure 1: Relationship between functions provided by application software S and functions used by a user $U(i)$.

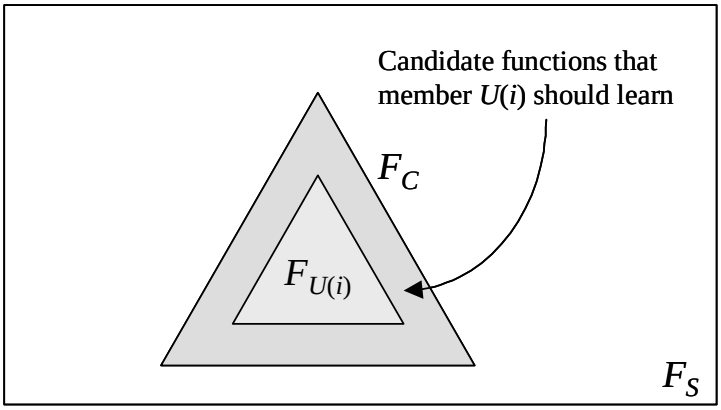


Figure 2: Relationship between functions used by a community *C* and by its member *U*(*i*).

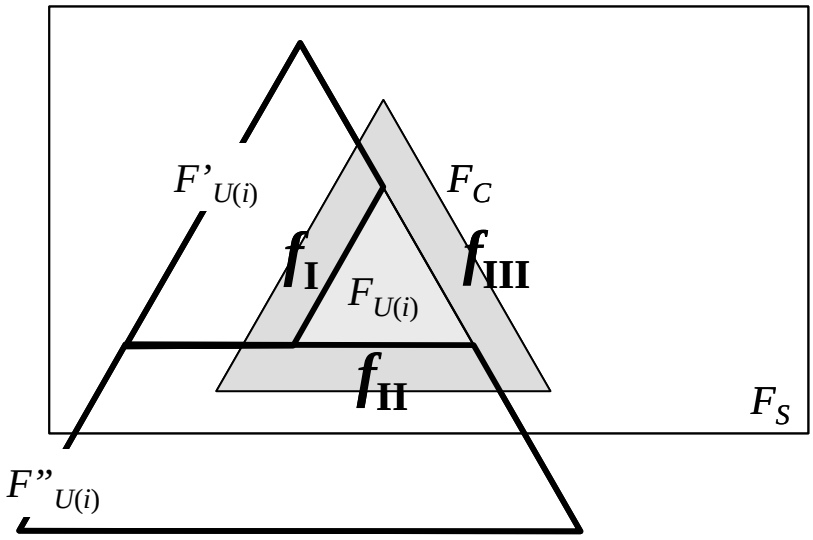


Figure 3: Three types of the candidate functions that the member *U*(*i*) should learn.

Type III: Function application software provides, but the user has not noticed.

By using a conventional online help system provided by application software, most of users could easily obtain usage knowledge of the *Type I* function. But they have to spend much time to identify the *Type II* function. Learning the *Type III* function is unpractical. An active help system, for example the "Did you Know" (DYK) system [9] or Microsoft's "Tip of the Day [6]," can provide users with usage knowledge of both *Type II* and *Type III* functions. However, the active help system needs a user model to identify these functions out of a lot of functions provided by application software. Developers have to spend much time to devise the user model for application software that is designed for diverse users.

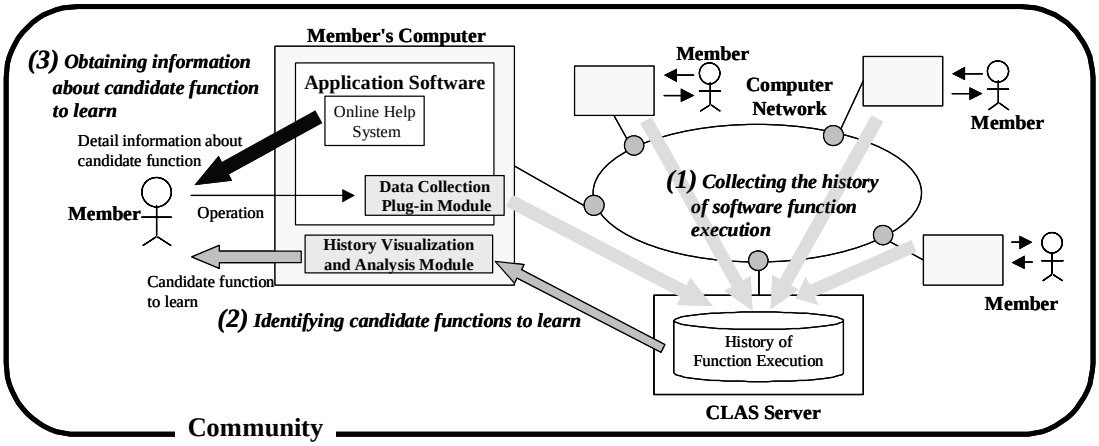


Figure 4: Three steps of learning process in the CLAS.

This paper describes an approach "Community-based Learning for application software (CLAS)" to let users learn useful functions more and more at low cost for the purpose of improving the users' productivity in using application software. Community members mutually compare their own histories of software function executions in order to identify a set of candidate functions that members should learn. A prototype implementation has been developed that automatically collects history of function execution on Microsoft Office 2000 applications.

CLAS has the following three advantages in reducing the learning cost of application software usage:

(1) Sharing usage knowledge in the form of data collected automatically

One of the best ways of obtaining knowledge of software usage is to ask an expert who knows everything about application software. But the asking to the expert costs much time and patience, and the productivity of the expert in using application software decreases. In addition, experts no longer exist in many of latest application software that provides users with numerous functions [2]. If the learning of application software usage is based on the asking to the other users, most users, except novices who have never used the application software, may be asked and disturbed by the others.

In the CLAS, the community members compare mutually their own histories of software function executions. The history of software function execution can be collected unobtrusively by our prototype system described in Section 3. Community members do not need to ask other members about application software usage.

(2) Simple model to identify functions to learn

Most of the conventional user support system, such as active help systems needs an

intricate model in order to analyze the user's behavior and identify functions that are related to user's task and that he/she should learn. But developing a user model for application software that is designed for diverse users is not easy. The user support system with an immature model discourages us from using it to learn the application software usage.

The CLAS uses very simple model to identify functions to learn. That is, the CLAS only assumes that among the functions that each member of a community uses, some differences in the software usage experiences of each member are observed and the function belonging to the difference is useful for the member to perform his/her current task. This assumption can be applied to almost all users of application software; and, application developer does not need to spend much time to devise a user model.

(3) Reinforcement of conventional online help system

The online help systems in many of the latest application software become large-scale as application software provides users with many functions. For example, in the case of Microsoft Word 2000, the size of the online help file is about 1.9MB. For most users, examining an entire description of online help is difficult. Recently, the online help system has the function to search the help description with question sentences or keywords. A novice user cannot even select a question sentence or a keyword. Only the experienced user uses the online help system originally devised for the novice user.

The CLAS does not take the place of a conventional online help system. The history of software function executions for members of a community is used only to enumerate the candidate functions to learn. The online help system gives detailed information about these candidate functions. The CLAS entirely uses the feature of the online help system and thus an initial cost to apply the CLAS to our environment is relatively small.

In what follows, in Section 2 a functional model and a learning process in the CLAS are discussed. Section 3 shows the current status of a prototype implementation designed to apply the CLAS to Microsoft Office 2000 applications. Section 4 describes an empirical evaluation of the CLAS. In the experiment, five subjects try to obtain usage knowledge of the new functions by using the prototype system. Section 5 introduces some related works. Section 6 summarizes the ideas discussed in this paper.

2 COMMUNITY-BASED LAERNING

2.1 Functional Model

Figure 1 shows the relationship between the functions provided by application software and the functions used by a user. $F_{U(i)}$ represents a set of functions that are regularly used by user $U(i)$. F_S represents a set of functions provided by application software S. The

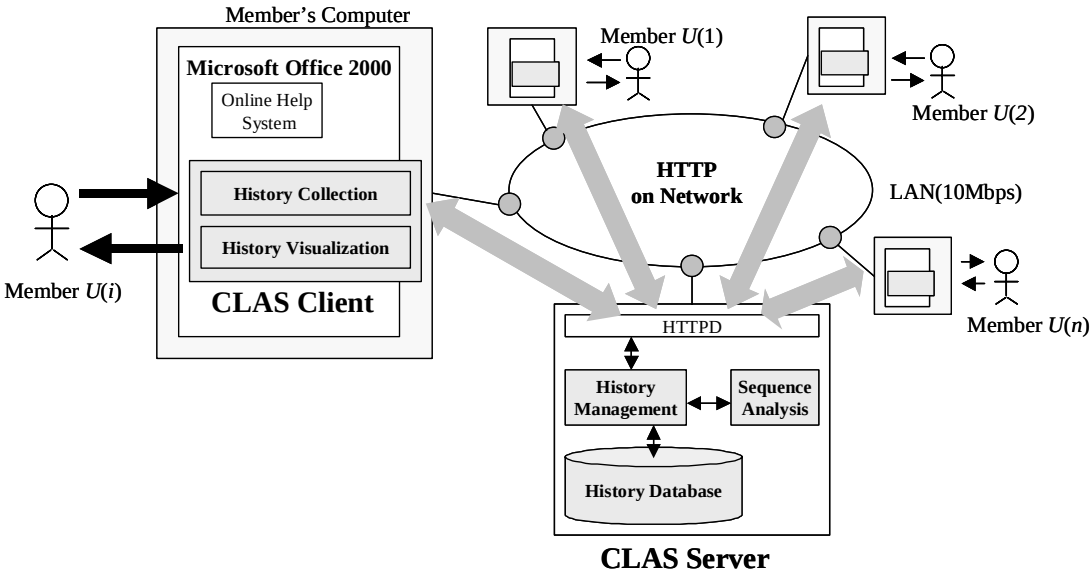


Figure 5: Prototype system overview.

objective of the CLAS is to increase $|F_{U(i)}|$ in order to improve the users' productivity in using application software.

Generally, the function usage rate ($|F_{U(i)}| / |F_S|$) is low and users need not care for additional existing functions if these functions are not relevant to their current tasks and/or interests. The learning support system has to select functions related to users' tasks and interests, out of many unknown functions provided by application software.

In the CLAS, we direct our attention to the function that the community member uses to select functions related to users' tasks and interests. Figure 2 shows the relationship between functions used by a community and by its members. The community consists of two or more members who use the same application software for the same purpose.

F_C represents a set of functions that are regularly used by some member of community C. That is, F_C is the union of the sets $F_{U(i)}$ (i = 1,…,n. n is the number of members of the community C) and $F_{U(i)} \subseteq F_C$. Moreover, set of candidate functions that the member $U(i)$ should learn is defined as a difference set $F_C - F_{U(i)}$. It is based on the assumption that among the functions that each member of a community uses, some differences in the software usage experiences of each member are observed and the function belonging to the difference is useful for the member to perform his/her current task.

In Figure 3, we classify the candidate functions that the member $U(i)$ should learn into the following three types as defined in [4].

Type I ($f_{\mathrm{I}} = F'_{U(i)} \cap (F_C - F_{U(i)})$): Functions that member $U(i)$ knows vaguely and uses only occasionally. $U(i)$ does not know the usage procedure of these functions and/or in which context they work effectively. Since $U(i)$ knows a concrete name and/or an outline of this type of function, he/she can easily obtain its information by using a online help system.

Type II ($f_{\mathrm{II}} = F''_{U(i)} \cap (F_C - F_{U(i)})$): Functions that the application software S provides and member $U(i)$ believes its existence in application software S, but are never actually used. If his/her guess about the name of a function or the keyword characterizing the function are related to the actual name or keyword, he/she can obtain information on this type of function by using the online help system.

Type III ($f_{\mathrm{III}} = (F'_{U(i)} \cup F''_{U(i)})^c \cap (F_C - F_{U(i)})$): Functions application software S provides, but member $U(i)$ has not noticed. Since $U(i)$ cannot select a question sentence and/or a keyword to search for unexpected functions, finding this type of function by an online help system is difficult.

Where $F'_{U(i)}$ represents the set of functions that are known vaguely and used only occasionally by member $U(i)$ of the community C, often requiring the online help systems ($F_{U(i)} \cap F'_{U(i)} = \phi$, $F'_{U(i)} \subseteq F_S$). $F''_{U(i)}$ represents a set of functions that member $U(i)$ believes its existence in the application software ($F_{U(i)} \cap F''_{U(i)} = \phi$, $F'_{U(i)} \cap F''_{U(i)} = \phi$).

2.2 Learning Process

The learning processes in the CLAS consist of the following three steps (See Figure 4):

Step 1: Collecting the history of software function execution

The history of software function execution for each member of the community is a primitive type of data of the CLAS. This history is time series of data containing at least the following three data: an ID of a function which be executed, an ID of the user who executed the function, and the time of the function execution.

"Data Collection Plug-in" on the computer of each member monitors function execution daily, in parallel to the execution of members' tasks on an application software. It also acts as a sender of data mentioned above to "CLAS Server." The CLAS Server accumulates the history of the function execution of all members as files.

Step 2: Identifying candidate functions to learn

Each member of the community can access to the CLAS Server any time. By comparing his/her own history of software function execution with other members' histories, the community member can easily identify candidate functions to learn. For example, the function that he/she has not used, but that most of the other members of the community

have used, is the leading candidate of a function that he/she should learn. In most cases, such an unexecuted function is classified into Type II or Type III, that is, such unexpected function is difficult to find by the online help system.

If he/she extracts subsequence of function execution from the history, including specific functions, he/she can find collocation among functions. Collocation represents one aspect of context of function use. The collocation that does not appear in his/her own history, but that appears in the history of most other members of the community may represent a new context of function use for him/her. In most cases, the function that has such collocation is classified into Type I. In other words, he/she vaguely knows it and uses it in a limited context.

Step 3: Obtaining information about candidate function

A member can obtain detailed information about the candidate function by executing it to trial or by the online help system. All of candidate functions may not be useful for him/her. If he/she decides a candidate function is not useful for his/her current task, he/she will quit obtaining detailed information about the candidate function.

3 PROTOTYPE SYSTEM

3.1 Overview

A prototype system is currently developed for applying the CLAS to Microsoft Office 2000 applications [7][10]. It provides functions for supporting the first two steps of learning process in the CLAS described in subsection 2.2. Using the online help system of Microsoft Office 2000 applications can perform the third step. The prototype system consists of the following two subsystems (See Figure 5):

(1) CLAS Client

It works on a community member's computer with Microsoft Office 2000 applications. Any community member can use it through user interface of the applications. CLAS Client provides members with functions to collect the history of function execution of a member on these applications, and to show the collected history and its analysis result. It is implemented as a set of plug-in modules (Microsoft COM add-in modules). Total size of these modules is 1,500 LOC in Visual Basic.

(2) CLAS Server

It works on a computer connected with the community member's computers in which CLAS Client works. CLAS Server provides members with functions to accumulate and maintain the history of function execution of all members of the community, and to extract subsequences of function execution from the history. It is implemented as a set of Java servelet. Total size of these modules is 7,000 LOC in Java.

These two subsystems communicate through computer network (LAN, Internet, etc.) according to Hyper Text Transfer Protocol (HTTP). Data to be exchanged between them are described in eXtensible Markup Language (XML).

3.2 Collecting the history of software function execution

"History Collection" module of CLAS Client monitors all operations corresponding to selection of menu item or short-cut button as function execution. At the time of the application closing, History Collection module adds the user ID and the application ID to these operations and then sends altogether to Hyper Text Transfer Protocol Daemon (HTTPD) on the CLAS Server (See Figure 6).

When HTTPD receives data from CLAS Client, it invokes "History Management" module and passes it. History Management module stores the received data as the history of function execution into a "History Database" in which history is classified by the user ID and by the application ID. Simultaneously, it invokes "Sequence Analysis" module.

Sequence Analysis module extracts all subsequences, whose length is less than seven, from the received data, and updates a "Subsequence Table" in History Database. Subsequence Table maintains all subsequences and their frequency counts. The reason for enumerating

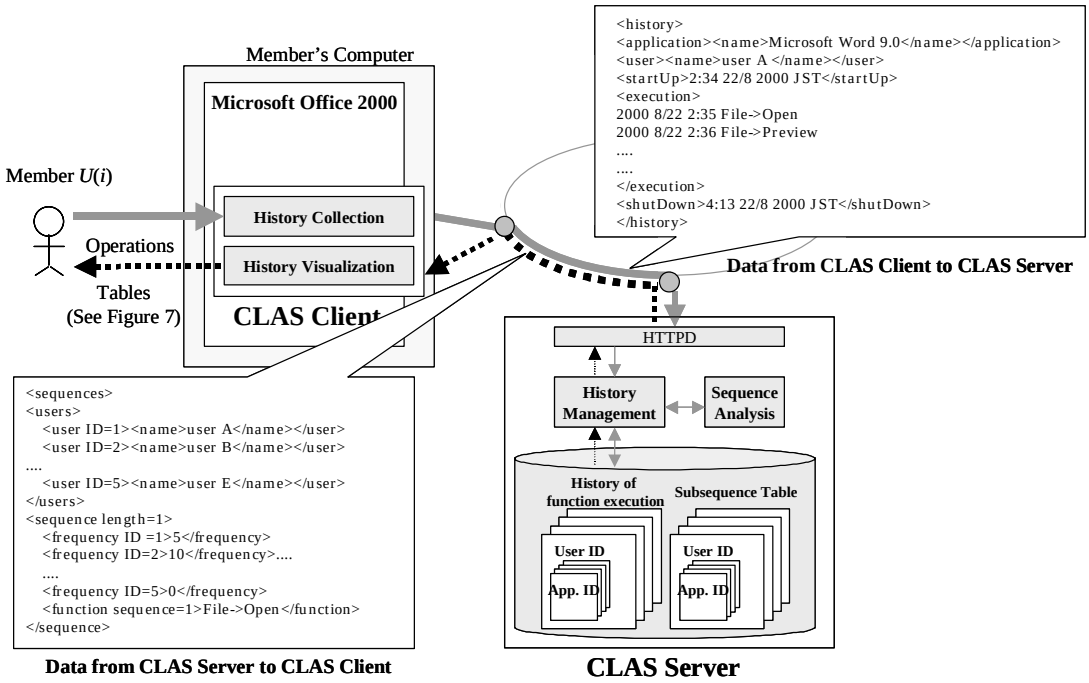


Figure 6: Data exchanging between two subsystems.

subsequences immediately after receiving data is that the extraction needs a lot of time.

3.3 Identifying candidate functions to learn

In response to the request of a community member *U*(*i*), History Visualization module requests the CLAS Server to send the data in History Database. If History Management module on the CLAS Server receives the request via HTTPD, it retrieves data, including the history of function execution and its subsequences, from History Database and requests HTTPD to send it to History Visualization module via computer network (See Figure 6).

History Visualization module processes the received data and provides member *U*(*i*) with the following two tables :

Table 1: Frequency count table for each function

This table is used for identifying the candidate functions to learn and obtaining keywords for retrieving their detailed information from the online help system. It consists of the following data items for each application function executed by the community members:

Execution rate by the other members (the ratio of the number of other members who executed the function, to the number of community members)

- Total number of frequency counts
- Frequency counts by each member
- Function name (Path from top menu in menu hierarchy or from tool bar)

Function of which “Execution rate by the other members” is nearly 100% and of which “Frequency count for *U*(*i*)” is nearly 0, is the candidate function *U*(*i*) should learn. If there are a lot of candidates, “Total number of frequency counts” can be used for prioritizing these candidates in learning.

Figure 7(a) shows an example of a screenshot of “Frequency count table for each function” in the prototype system. In this example, the community consists of five members (“User A”, “User B”, “User C”, “User D”, and “User E”). “User C” requested the prototype system to show this table. Each row represents function executed by some member of the community. The third, sixth, and eighth rows (functions) are candidate functions “User C” should learn. It is because those “Execution rate by the other members” are nearly 100% and those “Frequency count for User C” are nearly 0. In addition, the rows (functions) of which “Frequency count for User C” is zero are high lightened (colored and bold faced). These functions may be Type II or III functions that “User C” should learn.

Table 2: Frequency count table for each subsequence in Key Word in Context (KWIC) format

This table is used for clarifying collocation among functions for the purpose of finding

new context of Type I function usage. It consists of the following data items for each subsequence, including a function specified by member $U(i)$, found in histories of function execution of the community members:

- Execution rate by the other members
- Total number of frequency counts
- Frequency counts by each member
- Series of function name in subsequence

The last data item "Series of function name in subsequence" is represented in the form of *KWIC*, that is, a function specified by member $U(i)$, is located in center. If he/she clicks a cell of function name in subsequence, background color of all cells containing the same function name as the cell he/she clicks are changed.

Figure 7(b) show an example of a screenshot of "Frequency count table for each subsequence" in the prototype system. In this example, the community members are the same as Figure 7(a). "User C" requested the prototype system to show this table for seeing the usage context of the function "Index and Tables..." Each row represents the subsequence found in histories of function execution of the community members. Ten columns from right represent data item "Series of function name in subsequence" in the form of *KWIC*. "User C" clicked cells containing "Caption..." or "Style..." in order to investigate the collocation among these two functions and function "Index and Tables."

4 EXPERIMENT

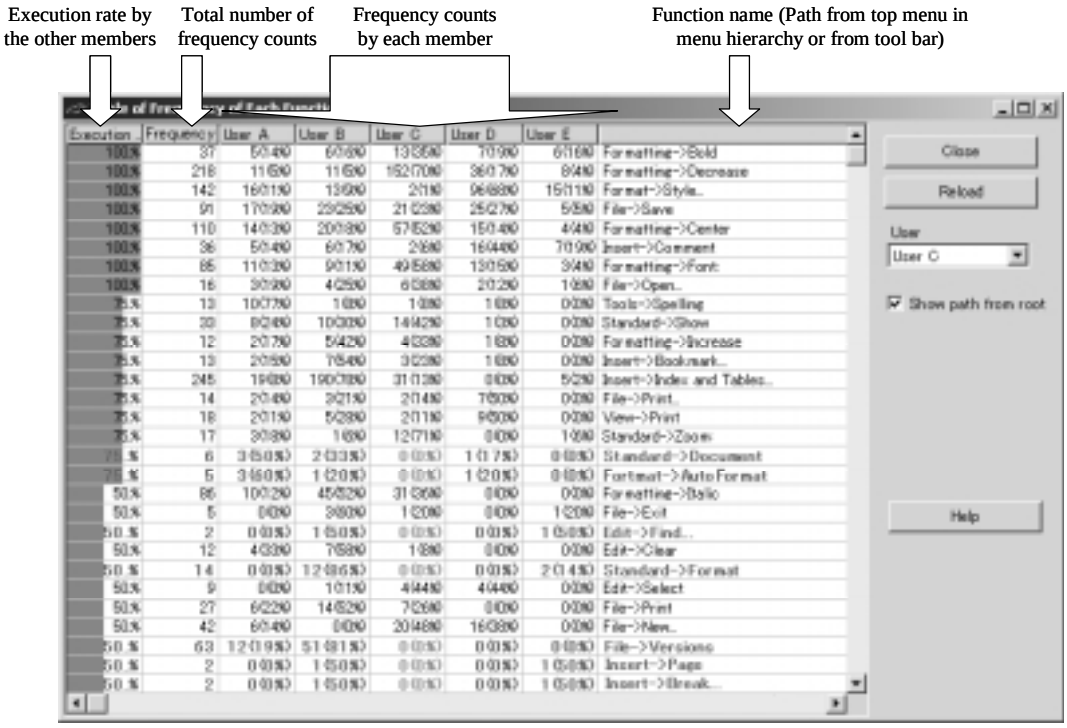
4.1 Outline

This section shows one of our experiments using the prototype system described in Section 3. The goal of the experiment was to evaluate the effectiveness of the proposed approach CLAS. The experiment is designed to make sure, by using the prototype system, that community members can identify candidate functions to learn, and obtain usage knowledge of new functions whose existences were unknown to them. We gave a same task on Microsoft Word 2000 to each subject. We want to see how many new functions subjects identify and use even in such small task.

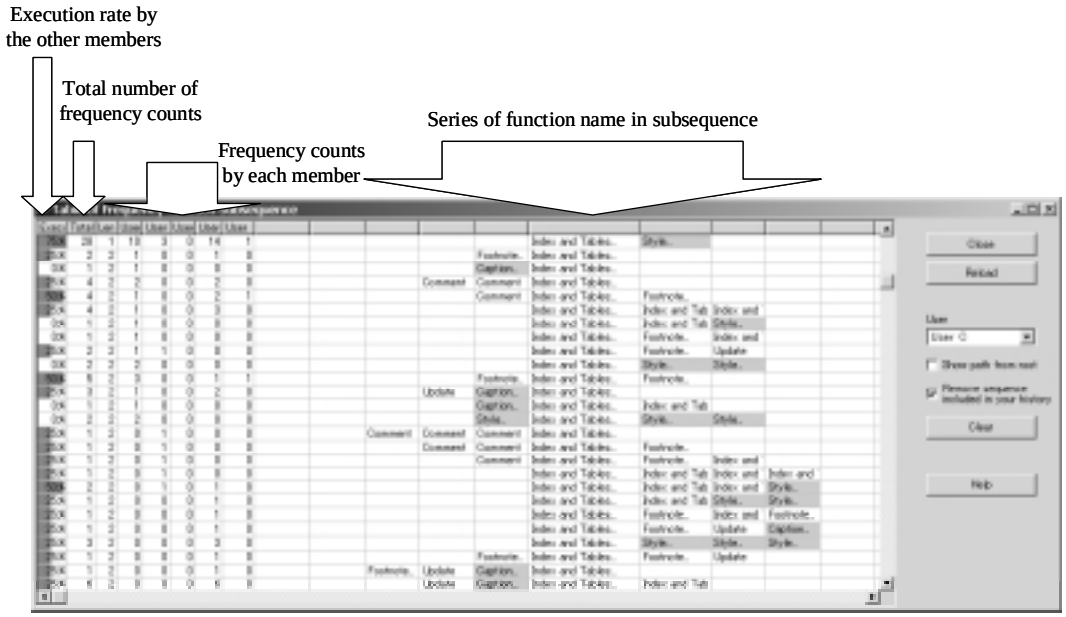
Important characteristics of the experiment were:

(C1) Five subjects ($S1$, $S2$, $S3$, $S4$ and $S5$) performed the same task and used the prototype system. Three of them are master course students and two others are professors of a software engineering laboratory at Nara Institute of Science and Technology.

(C2) The task is to create a Word document from a plain text. The plain text to be formatted was given to each subject in the form of an electronic file (See Figure 8(a)). The instruction

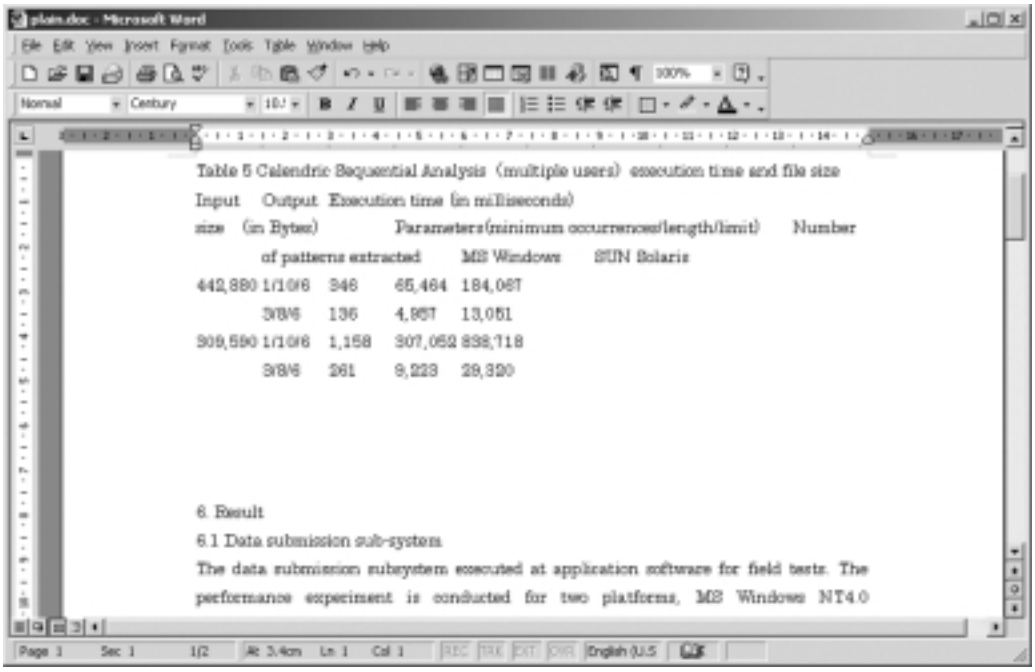


(a) Example of "Frequency counts table for each function"

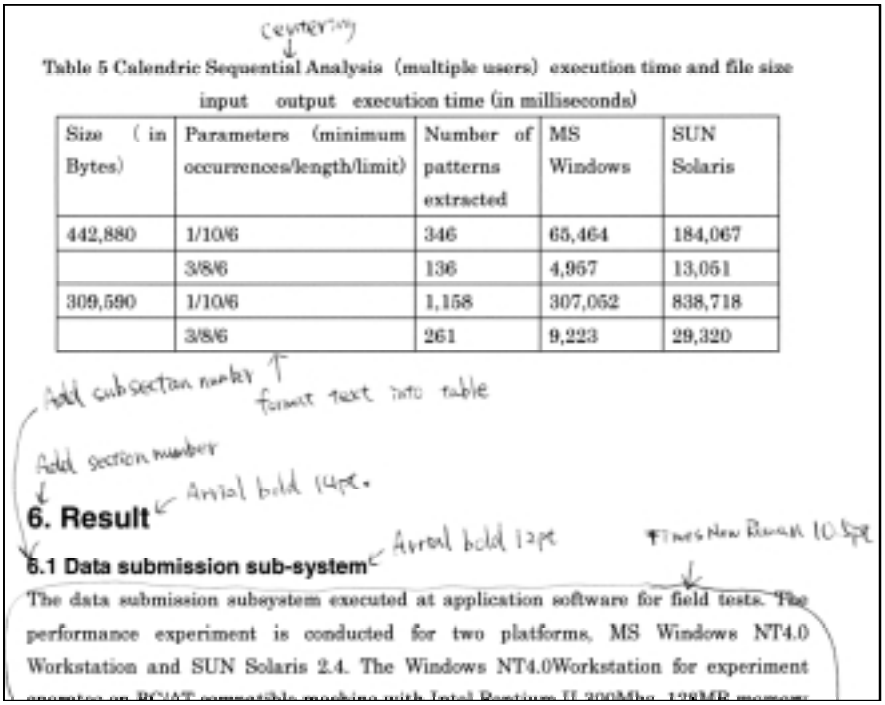


(b) Example of "Frequency counts table for each subsequence" in Key Word in Context (*KWIC*) format

Figure 7: Three tables provided by History Visualization module.



(a) Portion of plain text file to be formatted



(b) Portion of instructions to subject

Figure 8: Task assigned to each subject in the experiment

*T*5);

*T*1: Create a table of contents at the beginning of a document
*T*2: Add page number
*T*3: Convert text to table
*T*4: Change the font of titles of sections and subsections
*T*5: Number the titles of sections and subsections

(C3) Microsoft Word 2000 has the following five functions (*F*1, *F*2, *F*3, *F*4 and *F*5) that are very useful for doing those sub-tasks, respectively.

*F*1: Insert → Index and Tables
*F*2: Insert → Page Numbers
*F*3: Table → Convert → Text to Table
*F*4: Format → Style
*F*5: Format → Bullets and Numbering

Where "→" represents a path in menu hierarchy or tool bar on Microsoft Word 2000.

(C4) The history of function execution by each subject was collected by the prototype system. Subjects started to refer tables provided by the History Visualization module after thirty minutes from the start of the experiment.

(C5) After task completion, we had an interview with subjects in order to identify which set shown in Figure 3 those five functions (*F*1, *F*2, *F*3, *F*4 and *F*5) belong to.

4.2 Results

Table 1 shows the summary of experimental data. The most experienced subject *S*5 completed the task in the shortest time (47 minutes) with the fewest function executions (87). Subject *S*4 did not complete the task since he had no idea to add page number to his document (sub-task *T*2) and could not find any appropriate function to do that. Subject *S*4 executed the most functions (167) in the experiment. Subject *S*2 executed the fewest functions (21) and spent the longest time to complete the task, among four subjects who completed the task.

Table 2 shows how many new functions subjects discovered and used by the prototype system. Eight out of twenty-five (= five functions × five subjects) functions were well known to five subjects and used in the experiment, which belongs to the set $F_{U(i)}$. Other eight functions were not used in the experiment. We cannot identify to which set such unused functions belong, based on our collected data in the experiment.

The total number of functions learned by using the prototype system is eight. The total number of functions belonging to the sets f_{I}, f_{II}, and f_{III} is 1, 2, and 5, respectively. That is,

Table 1: Summary of experimental data

Subject ID	Application experience	Time of task completion (min.)	# of executed functions	# of function executions
*S*1	< 1 year	68	24	99
*S*2	2 years	82	21	134
*S*3	3 years	58	36	139
*S*4	3 years	> 82*	21	167
*S*5	> 5 years	47	27	87

* He could not complete task in the time limit.

Table 2: Function type

	*F*1	*F*2	*F*3	*F*4	*F*5
*S*1	*Not Used*	II	III	III	U
*S*2	II	U	U	U	III
*S*3	U	I [H]	*Not Used*	U	III
*S*4	*Not Used*	*Not Used*	*Not Used*	*Not Used*	*Not Used*
*S*5	I	U	III	U	*Not Used*

I: Function belonging to the set f_{I}. II: Function belonging to the set f_{II}.
III: Function belonging to the set f_{III}. U: Function belonging to the set $F_{U(i)}$.
[H]: Function of which usage knowledge was obtained only by the online help system.

the subjects using the prototype system could easily learn about a third of the functions, and more than half of these functions were new functions whose existences were unknown to them. On the other hand, the function, learned only by using the online help system in Microsoft Word 2000, without using the prototype system was only one.

In addition, all five functions were well known to at least one subject and were learned and used by at least one other subject during the task. All subjects who had completed the task could obtain usage knowledge of at least one function by the prototype system. It is hard to consider that the characteristic of subjects or tasks has greatly influenced the result.

Although it must be discreet to generalize the experiment results immediately, we claim that the proposed approach CLAS has high possibility of supporting learning of application software functions.

4.3 Lessons Learned

Lessons learned from the experiment include the following:

(L1) For novice user, the CLAS is very useful for pushing up his/her own productivity in using the application software into a certain level. Novice (*S*1) could easily identify candidate functions to learn by using the prototype system. He considered that three of the candidate functions are useful for doing his task. He obtained usage knowledge by using the online help and then used them in his task. By using these three functions, his productivity increased during the experiment. Total time to complete the task was shorter than two other subjects (*S*2 and *S*4) who had more usage experience of Microsoft Word 200 than *S*1.

(L2) For experienced user, the CLAS is very useful for finding unaware functions of application software. Experienced subject (*S*5) also obtain usage knowledge of two new functions by using the prototype system. The existence of one of them was unknown to him. That is, by performing a task for only one hour, he found a new function that he has never been aware of in the past five years.

(L3) Data item "Function name with path from top menu in menu hierarchy or from tool bar" in tables (See Figure 7) provided by History Visualization module provides subjects with useful information for obtaining detailed information of the candidate functions. Most of subjects said that word in the data item is appropriate keyword to retrieve detailed information of the candidate functions from the online help system. They can easily find and actually execute the candidate function in the menu system or tool bar of the application software by following the path, if necessary.

(L4) When we design the task for the CLAS, we have to take into account the skill level of members to the application software, average time needed to complete the task, and so on. The task assigned to each subject was too heavy for subject *S*4. Such heavy task may discourage users from obtaining new knowledge of the software functions, and the CLAS does not work properly.

(L5) We have to devise a mechanism of pre-processing history data to show it to users in concise and/or understandable way. Subject *S*4 could not handle a large amount of the history of function execution provided by the prototype system.

5 RELATED WORKS

5.1 User Support based on Operations in Application

In the area of predictive interface, a lot of researchers have proposed techniques for using the history of a user operation on application software for ease of use [2][3][8]. Masui et al. proposes the "Dynamic Macro Creation [8]." With the proposed technique, when a user types a special "repeat" key after doing repetitive operations in a text editor, an editing

sequence corresponding to one iteration is detected, defined as a macro, and executed at the same time. It is effectively works for simple repetitive operations or tasks, but cannot help users to obtain knowledge of unknown operations (or functions). The current version of Microsoft Office 2000 applications shows full menus after short delay, if we use its default setting. User can select the menu item easily and perform effectively his/her routine work, but he/she may lose an opportunity to see unused functions. These techniques cannot help users to avoid to get stuck on sub optimal plateaus, unless they use the history of operations of the other users.

5.2 Application Usage Monitoring via Network

Several researchers have proposed techniques for application usage monitoring via computer network. Hilbert et al. proposes an approach and prototype system that makes large-scale collection of usage data over the Internet a practical possibility [5]. In their approaches, expectations of application developers about how application users will behave are encoded in the form of agents. The agent monitors application usage and notifies the violation of the prescribed expectations to the developer via the Internet. The developer can obtain large amount of usage data in effective way, but the user directly obtains no feedback. In addition, the developer obtains no usage data if the user does not violate expectations prescribed by him/her. The developer may test the validity of their application in user environment all over the world, but he/she has to enumerate a lot of expectations in order to catch diverse behaviors by diverse users.

6 CONCLUSION

This paper proposed a new concept “Community-based Learning to application software (CLAS),” and showed the potential capability of it by conducting experiment with the prototype system. Authors and our colleagues are continuously using our prototype system in our daily work. The system collects and accumulates the history of function execution of all of us in unobtrusive way. Even if no special task is assigned to all members of the community, it is possible to identify a candidate functions by comparison of the accumulated history. We need only five minutes to compare our own history with ones of the other members and to identify candidate function to learn. The CLAS is capable of developing knowledge of software usage and sharing it among the users in systematic and effective way.

ACKNOWLEDGEMENTS

This study is financially supported by the Proposal-based New Industry Creative Type Technology R&D Promotion Program from the New Energy and Industrial Technology Development Organization (NEDO) of Japan.

REFERENCES

1. Carroll, J.M., and Rosson, M. B.: “Paradox of the Active User, Interfacing Thought: Cognitive Aspect of Human-Computer Interaction,” MIT Press (1987).

2. Cyper, A.: “Eager: Programming repetitive tasks by example," *Proc. of the ACM Conference on Human Factors in Computing Systems*, (April 1991), 33-39

3. Darragh, J. J., Witten, I. H., and James, M. L.: “The reactive keyboard: A predictive typing aid," *IEEE Computer*, 23, 11 (1990), 41-49.

4. Fischer G.: “User modeling: The long and winding road,” *Proc. UM99 (User Modeling) Conference* (Banff, Canada, June 1999), Springer Verlag Wien New York, 349-355.

5. Hilbert, D. M., and Redmiles, D. F.: “An approach to large-scale collection of application usage data over the Internet,” *Proc. 20th International Conference on Software Engineering*, (1998), 136-145.

6. Horvitz, E.: “Agents with beliefs: Reflections on Bayesian methods for user modeling,” *Proc. UM97 (User Modeling) Conference*, (Sardegna, Italy, 1997), Springer Verlag Wien New York, 441-442.

7. Morisaki, S., Monden, A., Matsumoto, K., Inoue, K., and Torii, K.: “Two-level user interface evolution by sharing usage logs,” *Proc. of International Workshop on Principles of Software Evolution'99* (July 1999), 70-73.

8. Masui, T., and Nakayama, K.: “Repeat and predict – Two keys to efficient text editing,” *Proc. of ACM Conference on Human Factors in Computing Systems* (1994), 118-123.

9. Owen, D.: “Answers First, Then Questions,” In Norman, D.A. and Draper, S.W. (Eds.), *User-Centered System Design, New Perspectives on Human-Computer Interaction*, (1986), Lawrence Erlbaum Associates, Inc., Hillsdale, NJ, 361-375.

10. Torii, K. Matsumoto, K., Nakakoji, K., Takada, Y., Takada S., and Shima K.: “Ginger2: An environment for CAESE (Computer-Aided Empirical Software Engineering),” *IEEE Transactions on Software Engineering*, 25, 4 (1999), 474-492.