Computer Analysis of User Interfaces Based on Repetition in Transcripts of User Sessions

ANTONIO C. SIOCHI
Christopher Newport College
and
ROGER W. EHRICH
Virginia Tech

It is generally acknowledged that the production of quality user interfaces requires a thorough understanding of the user and that this involves evaluating the interface by observing the user working with the system, or by performing human factors experiments. Such methods traditionally involve the use of videotape, protocol analysis, critical incident analysis, etc. These methods require time consuming analyses and may be invasive. In addition, the data obtained through such methods represent a relatively small portion of the use of a system. An alternative approach is to record all user input and system output (i.e., log the user session). Such transcripts can be collected automatically and noninvasively over a long period of time. Unfortunately this produces voluminous amounts of data. There is therefore a need for tools and techniques that allow an evaluator to identify potential performance and usability problems from such data. It is hypothesized that repetition of user actions is an important indicator of potential user interface problems.

This research reports on the use of the repetition indicator as a means of studying user session transcripts in the evaluation of user interfaces. The paper discusses the interactive tool constructed, the results of an extensive application of the technique in the evaluation of a large image-processing system, and extensions and refinements to the technique. Evidence suggests that the hypothesis is justified and that such a technique is convincingly useful.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*evaluation / methodology*

General Terms: Human Factors, Measurement

Additional Key Words and Phrases: Maximal repeating patterns, repeated usage patterns, transcript analysis, usability, user interface evaluation, user interface management systems

This research was partially supported with funds obtained from grants by the IBM Corporation, Software Productivity Consortium and the Virginia Center for Innovative Technology. We also acknowledge the support of the Dialogue Management Project and the Contel Technology Center for funding current research on this topic.

Authors' addresses: A. Siochi, Department of Physics and Computer Science, Christopher Newport College, Newport News, VA, 23606, email: siochi@pcs.cnc.edu; R. W. Ehrich, Department of Computer Science, Virginia Tech, Blacksburg, VA 24061-0106, email: ehrich@vtcs1.cs.vt.edu.

1. INTRODUCTION

Recently, support has been growing for the process of iterative refinement and rapid prototyping—a development process in which user feedback is an essential element [5, 11, 13, 25, 26, 27]. If one accepts the arguments in favor of such a methodology, then one is faced with the task of empirical evaluation and the problem of locating interface weaknesses as quickly and inexpensively as possible. Apart from informal observation, most evaluation involves either formal experiments or field research.

Formal experiments utilize controls such as benchmark tasks in order to ensure statistically significant results. Videotape is normally used to record the experiment for subsequent analysis. This step is often tedious and time-consuming, taking up to a day or more for a one hour experimental session [17]. Apart from this problem, a more serious drawback is the lack of a natural work context imposed by the benchmark tasks: "... the deliberate restricting of focus ..., has the effect of making an a priori value judgment that these operations are at the heart of usability for the system" [25]. Hence the alternate technique of field research involves visits to the users' place of work, where evaluators interview users *as they work*. Unfortunately, this technique examines only a thin slice of the user's daily experience with the system. Eason [7] also points out that "It is very easy to overlook usability problems in subjective evaluations."

Formal experimental techniques tend to provide very exact results, but about narrow and specific areas. *However, there are few formal procedures for identifying those interface aspects that require experimental investigation.* In addition, experiments are typically expensive to conduct, and they can take a long time. Field research techniques can provide results in less time and with less money than formal experiments; however, the results can not be validated in the strict experimental sense. Both the formal experimental techniques and field research techniques are invasive as well, although in different ways: the experimental techniques pluck users out of their natural environments, whereas the field techniques insert the researcher into that environment.

The contribution of this research is a new, relatively low-cost evaluation technique for helping the developer locate interface problems. It is based upon the detection of repeated user actions in computer-collected transcripts of user sessions. The algorithm and tools developed to detect this repetition can rapidly identify sections of transcript associated with potential interface problems. Because of this capability, there is no need to review all the raw data. This results in faster analyses and makes feasible the analysis of data collected over extended periods of time.

2. TRANSCRIPT ANALYSIS

One empirical technique that preserves portions of the natural work context and is not invasive is *transcript analysis*. All user input and system output is captured in real-time to a file which is analyzed later. This file, or transcript,

serves the same role as a videotape of the terminal screen. Both are records of the interactions performed in a user session.

Transcript analysis has several advantages. First, the data represent the user's interaction with the system under actual working conditions, rather than on contrived laboratory exercises. Second, since the data are stored in on-line files, they are accessible to algorithmic analysis and data reduction techniques. Such analytic tools can relieve the evaluator of the tedium associated with analysis, thereby encouraging the use of evaluation. Third, because data collection is automatic, there is no need for an observer or experimenter to be present. This absolutely eliminates any interference due to an observer, and means that data can be collected *in situ*, rather than at a laboratory, and collected from many users at the same time. Fourth, analytic tools also enable rapid analysis, thus providing quick feedback to designers or immediate debriefing of subjects. The analytic tools also make possible analyses involving vast quantities of data, which would never have been considered with videotape.

Transcript analysis also has some intrinsic limitations. First, since the data represent actual work contexts and not any predefined tasks, it is difficult to infer from the transcripts what a user was trying to do. This is especially true when only user input is present in the transcript. However, it is also possible to conduct short, focused interviews with users to discuss portions of the transcript identified by analysis.

Second, users tend to use a subset of a software application rather than the entire system, with the result that transcripts do not provide data for the entire system. It is important to realize, however, that it is also useful to know which parts of a system are not used.

Similarly, some users exercising system features experience no identifiable problems while others experience difficulties with the same features. Thus it is important to collect data from as wide and diverse a set of target users as possible.

Third, some version of the system must be implemented and installed in order for transcripts to be collected. Thus design changes suggested by the transcript analysis can only be applied to the next version of the system. This limitation does suggest that transcript analysis may not be appropriate for early evaluation of systems. However, once some version of the system or its prototype is implemented this technique is readily applicable.

Finally, although neither videotape nor transcripts capture user intent, transcripts capture less information than videotape. For example, transcripts do not capture the physical work environment or visual information related to the user tasks. The question, however, is whether logging user sessions can capture enough data to be useful. In addition, it is essential that the analysis be automated in some fashion, since merely recording user input replaces hours of videotape with megabytes of files.

This problem can be seen in the approach of Neal and Simons [18]. They set up one computer to intercept, record, and time-stamp each keystroke. Each keystroke was then passed to the other computer which ran the application system, so that no modifications to the application system were required. Analysis was performed by "replaying" the keystroke file, that is, the file

was used like a videotape. Unlike videotape, the observer was able to annotate the logfile directly with observations (e.g., critical incidents) or comments. The system also provided some analysis help in the form of frequency of occurrence of critical incidents, the time between such incidents, or an incident and the next user keystroke, frequency of use of commands or function keys, time spent in help, total session time, and number of help requests. Neal and Simons found their methodology to be " . . . very effective for objective evaluation and comparison of software including the user interface design and software documentation." However, this still involves manual review of the entire transcript.

There are several examples of automating the analysis. Each involves reduction to summary data. Cohill and Ehrich [6] describe a set of programs and routines developed to collect keystrokes and state information, and compress the raw data. The variables they measured were time spent in help, number of times help was invoked, and frequency of command use. They inserted calls to the transcript recording routines at strategic points in the code of the system they were investigating, then using their tools, reduced the resulting data to a form suitable for immediate input to SAS [20]. They found the tools they developed to be "extremely convenient," and report that it was better to collect as much data as needed, and reduce that data, rather than skimp on data collection despite the large amounts of data involved.

Good [10] analyzed existing keystroke data collected at numerous sites, representing the use of five different text editors. The results were used in the design of a new text editor. Transition frequencies between keystrokes were used to aid in the layout of keys, for example, the inverted-T layout of the cursor keys. The new command set was based upon command usage frequencies. The new editor itself was instrumented to log commands, and usage data were collected to determine the judiciousness of design decisions taken and to provide feedback for the next design iteration.

Hanson et al. [12] collected command usage data on UNIX [TM] and applied elegant statistical analysis techniques to the data. They determined a core set of commands by studying command frequencies. They also constructed a command transition matrix from the data. From this matrix and by applying multivariate grouping techniques, they were able to determine the degree to which a command is used together with many other commands. Treating this same matrix as a contingency matrix enabled them to determine the statistical sequential dependencies among commands. They prescribed some restructuring of the UNIX interface based upon these results.

One problem shared by these techniques is data collection. The collection methods involved ad hoc interface modification or external keystroke collection hardware. Ideally, transcript analysis should be supported by a User Interface Management System (UIMS) [8, 13, 19]. This would eliminate the need for interface modifications to collect user input and provide integrated support for an evaluator's data collection, management and analysis activities. At least two UIMSs have supported transcript collection [8, 19]. In

[TM] UNIX is a trademark of Bell Laboratories.

particular, Olsen and Halversen [19] provided metrics such as performance time, mouse motion, command frequency, command pair frequency, number of physical and logical input events, and visual and physical device swapping. The interface profile used by their UIMS to generate the interface was also used by the metrics computation and report generation program to produce a list of commands ranked from worst to best for each metric.

The examples presented above all involve reduction of data to summary results, such as command usage frequencies, command or keystroke transition frequencies, or time spent in a certain state. *Such measures are convenient but do not completely reflect the kind of information a person can detect by reading transcripts.* For example, when one reads transcripts, one notices patterns of user commands such as repeated sequences of a command interspersed with error messages, or a group of commands that occurs at more than one place in the transcript. The human evaluator is naturally drawn toward these patterns or repeated sequences of commands.

One could search for specific patterns of user actions, such as a sequence of three help commands, but this assumes that such patterns are the only important indicators of interface problems. On the other hand, making no assumptions at all requires reading entire transcripts. The technique described in this paper attempts to locate problems by detecting repetition rather than searching for specific patterns. While other evaluation methods may be used to test small interfaces, in very large interfaces it is extremely difficult to test the interface or even make good hypotheses upon which to base limited tests. What is usually done is to ask users to take notes and comment on problems they encounter. It has been our experience that this does not work too well—users seldom respond to requests for problems. Our method generates reasonable hypotheses from massive user transcript data files.

3. MAXIMAL REPEATING PATTERNS (MRPs)

A user session transcript is the complete record of user input actions and system responses generated as a result of using the system. User input actions are extracted from this transcript and represent the time-ordered sequence of user inputs. In a command line interface this extract consists of lines of command strings. Other interface styles, such as direct manipulation [21], will have different types of user actions. Such actions, however, can be represented in some textual fashion [23] and thus should yield to the techniques described here. For example, mouse clicks could be recorded as screen coordinates, as selections of screen objects or as menu selections depending upon the grain of analysis desired. Since the MRP algorithm examines only user inputs, representing graphical output (a more difficult problem) is not strictly necessary to test this evaluation technique. It is therefore sufficient to examine only command line interfaces as a preliminary investigation.

3.1 Repeating Patterns: The Repetition Hypothesis

This paper hypothesizes that repeated sequences of user actions can indicate problems with the user interface. The Rationality Principle [4] is assumed, that is, users carry out a sequence of tasks to achieve some goal. Users

accomplish tasks by manipulating the computer's input devices and monitoring its output devices in a manner dictated by the user interface. Commands and data are entered and the resulting output is observed. If the results are satisfactory then the task was accomplished. If the results are not satisfactory, or if an error occurs, users must respond since the task was not accomplished. Transcripts of user sessions are records of this flow of input, resulting output and input in response. Therefore, it can be reasonably assumed that such transcripts reflect sequences of tasks users carry out. *Furthermore, inasmuch as these sequences of actions are made possible, and are partly governed by the interface, the quality of the user's interaction with the system is also reflected by those transcripts.*

Over a sufficiently long period of time, a user is likely to repeat those tasks which are useful or which support that user's computational needs. Otherwise those tasks were not useful or the user has constantly changing needs. Since these tasks are accomplished by issuing commands, then over that period of time it is reasonable to expect repetition of command sequences in a user transcript. Moreover, when users have problems using the system, they are also likely to repeat sequences of commands in the attempts to accomplish their task. Consider the following typical transcript of a UNIX session:

```
% cat garam.masala
garam masala. No such file or directory
% cat garram masala
garram.masala: No such file or directory
% cat garamasala
garamasala  No such file or directory
% ls
README spices/
% ls cpises
cpises: No such file or directory
% ls spices
GARLIC  chillies  cloves  cumin  tumeric
basil  cinnamon  coriander  garam.masala
% cat garam masala
garam masala  No such file or directory
% cd spices
% cat garam.masala
  cinnamon, cloves, and other good stuff
%
```

In this example, the user is trying to print the contents of the garam masala file. The user attempts this task three times, each time receiving an error message. Finally, the user decides to list the contents of the current directory believing that the error is a misspelled filename. The listing reveals instead that the error was the issuing of the cat command in the wrong directory. As a result, the user searches for the correct directory by repeating the ls command.

Another example is that of frequent similar dialogues with the help system. A command sequence such as

```
A >  TYPE B:INPUT*.TXT
ERROR. BAD FILENAME
```

```
A > HELP TYPE
/* help information for the type command is printed */
A > TYPE B:\INPUT*.TXT
ERROR: BAD FILENAME
A > HELP TYPE
```

might indicate a problem with the help entry for the TYPE command.

Thus we see that repetition of command sequences can be expected both in the routine, error-free user sessions and in the sessions where users have problems. Repeated action sequences may therefore mean any of the following:

(a) commands are at too low a level for the tasks,
(b) the user is circumventing an application malfunction,
(c) a user is searching for a way to do something, or
(d) some malfunction is causing loss of state information which must be regenerated

Case (a) would be indicated by repetition in error-free sequences. By detecting such frequently repeating actions and providing macros for them, it may be possible to reduce performance times and errors. Cases (b), (c), and (d) involve errors, clearly situations which an evaluator wants to see. In the UNIX transcript above, the repetition indicated a loss of state information (the current working directory) on the part of the user. The Results section lists examples of these cases.

3.2 The String Model of Transcripts

The problem of detecting repeating sequences of command strings in the extract is equivalent to detecting repeating sequences of characters in a string where each character in the string represents a command. The complete string then represents the entire extract and a task would be represented by some substring. Only substrings of length at least two will be considered.

Detecting repeating substrings presents some difficulties. Consider the string "abcabc." The repeating substrings are "abc," "ab," and "bc." Which repeating substrings should be reported? Recalling that each character in the string really represents some user action, the question is really "which substring or pattern is behaviorally interesting?"

Because "ab" and "bc" are substrings of the repeating substring "abc," it is more efficient to report just the substring "abc," since any substring of a repeating substring must also repeat. Apart from the computational expense of finding and reporting all repeating substrings, there is the expense of analyzing the prodigious volume of data that would be produced.

Consider further the string "abababab." Is the user performing 4 sets of "ab," or 3 sets of "aba" or "bab," or 2 sets of "abab," etc.? This question cannot be answered on the basis of syntax alone, but requires knowledge of the semantics of "a" and "b." By reporting only longest repeating substrings, "ababab" in this case, it is possible for the analyst to study substrings as

required and not be inundated with data. This research introduces a modification of these longest repeating substrings: *maximal repeating patterns*. For a formal definition, see Siochi [22].

3.3 Definition

A repeating pattern is a substring which occurs at more than one position in a string. A maximal repeating pattern, MRP, is a repeating pattern that is as long as possible, or is an independently occurring substring of a longer pattern. For example, in the string "abcdyabcdxabce," the substrings "ab," "abc," "abcd," "bc," "bcd," and "cd" are repeating patterns, whereas only "abcd" and "abc" are maximal repeating patterns. "abcd" is an MRP because it is not a substring of any repeating pattern, that is, it is of maximal length. "abc" is an MRP even though it is a substring of "abcd" because "abc" also appears independently of "abcd" (after "x"). This special case is an attempt to preserve "context." For example, "abc" occurs in two contexts, "abc**d**" and "abc**e**." It may be important to know that in one case "abc" precedes "d" while in another it precedes "e." MRPs may also overlap, as in "abcabcabc," where "abcabc" is the MRP (one instance occurs at the first position and the other instance at the fourth position).

3.4 Algorithm

The detection of MRPs and the positions at which they occur can be a very expensive task since the problem is not pattern search, but pattern detection (i.e., patterns are not known beforehand). A brute-force algorithm would exhibit an $O(n^3)$ time complexity, which for large n could discourage interface evaluators from using the method.

Siochi [22] has developed an order $O(n^2)$ algorithm based upon position trees [2, 24] to detect all the MRPs in a transcript, as well as report the positions at which they occur. Other algorithms exist which find longest repeated substrings (e.g., see Blumer et al. [3], Karp et al. [15] and Weiner [24]); however these do not detect the independently occurring patterns.

The MRP detection algorithm makes use of position trees [2, 20], which are trees whose leaves correspond exactly to each position in a string and whose arcs are labelled with characters of the string (see Figure 1). Each position in a string is uniquely identified by a substring—the shortest substring which occurs at that position and nowhere else. The path from the root to a leaf corresponds to this identifying substring. As a result, any proper prefix of this path is a repeating substring and occurs at the positions represented by the leaves in the subtree whose root is the terminal node in that path. For example, in Figure 1 the substring "bc" occurs at positions 1 and 4.

The algorithm starts with a longest repeating substring, since such a substring is an MRP by definition, and eliminates leaves from the position tree whose positions fall within that longest repeating substring. Referring again to Figure 1, the algorithm would start at the node labelled α. The prefix "abc" occurs at positions 0 and 3 and therefore any substrings occurring at positions 1, 2, 4, and 5 cannot be MRPs. The algorithm thus deletes

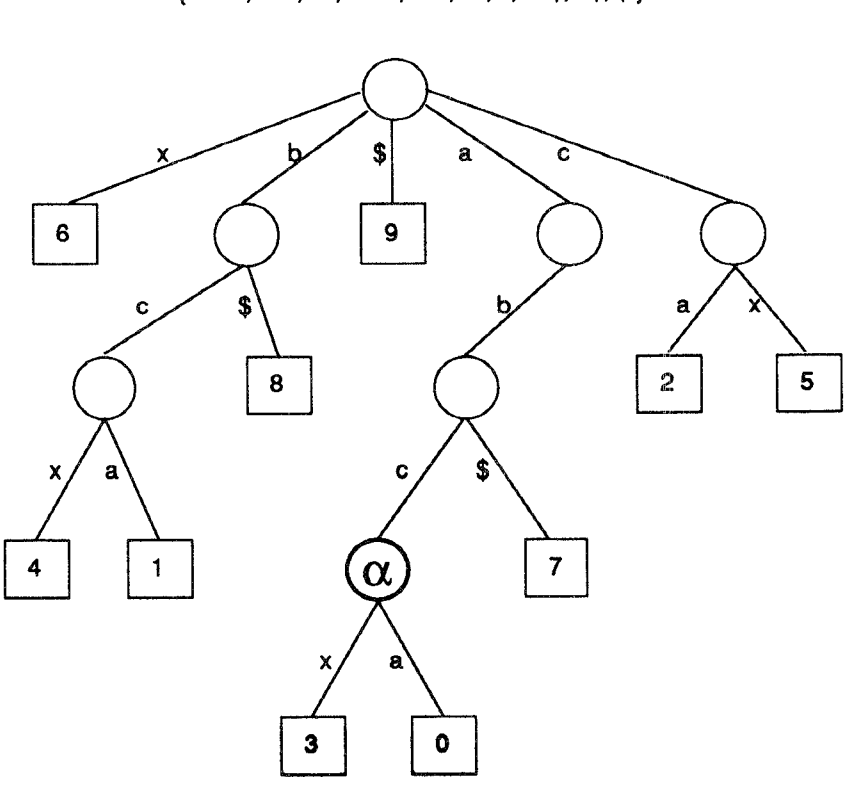

Fig 1. Position tree for the string S = "abcabcxab".

those leaves. This procedure is repeated for the next longest repeating substring, until no more leaves can be eliminated. The longest prefixes of the paths to the remaining leaves are the MRPs and the remaining leaves are the positions at which they occur.

4. TOOLS

4.1 The Normalizer

The normalizer translates raw transcripts into a standard form that the MRP tool uses, thereby keeping the MRP tool independent of the formats which data logging routines might use. As a result, transcripts of any system can be analyzed without changing the MRP tool, provided a normalizer can be written for that system's transcripts.

The normalizer consists of a couple of short AWK programs and a C-shell script. AWK [1] is a pattern scanning and processing language available on

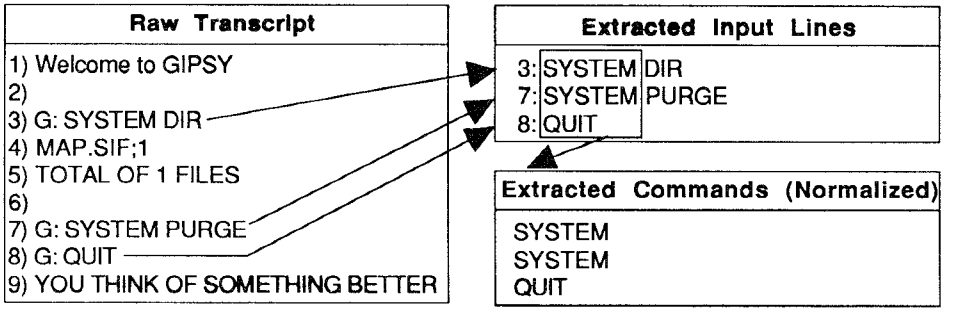

Fig. 2 The normalizer converts raw transcripts to a standard form

most UNIX systems, while C-shell is a UNIX command line interpreter. An AWK program is a sequence of pattern-action pairs. The AWK interpreter reads a line of the input file and executes the actions for each pattern matched by the input line.

Figure 2 shows the transformations carried out by the AWK programs. The first AWK program extracts user input lines from the raw transcript, including line numbers. The second AWK program then extracts the command portions which form the input for the MRP tool.

A more powerful means of building a normalizer would be to use LEX and YACC [14, 16], a pair of compiler writing tools available on UNIX systems. These tools transform a grammar describing the raw transcript file into a normalizer.

4.2 The MRP Tool

Capabilities. The MRP tool produces a list of MRPs from a normalized transcript. An evaluator can then

—scan this list,
—select MRPs from this list based on a few criteria (e.g, the length of an MRP),
—examine the MRPs at various levels of detail,
—obtain summary information about the transcript (e.g., number of MRPs found, frequency of occurrence of commands), and
—save any of this information to a file.

as well as perform operating system commands (in this case, UNIX) from within the tool. There is also a limited form of macro capability, which when combined with the C-shell allows a set of MRP tool operations to be performed automatically on several different normalized transcripts.

The tool does not produce metrics-style numbers indicating the usability of an interface; such metrics are an open research issue. It is important to realize that the MRP tool is not a "data summarizer," but an identifier of potentially interesting episodes in the transcript. The tool is valuable in two

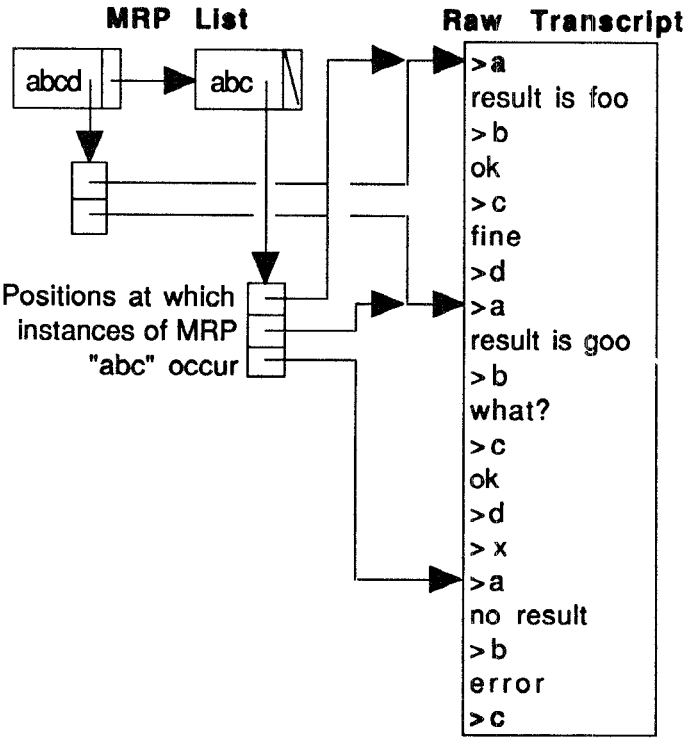

Fig. 3. Relationship of MRP list to the raw transcript file.

ways: an evaluator does not have to read the entire transcript to find repeating patterns and the tool may uncover patterns the evaluator might miss. It remains the evaluator's job to determine the significance of individual MRPs.

General operation. The MRP tool uses three types of input files:

.raw transcript data file containing both user input and system output,

.inp derived from .raw file; contains only user input lines from the .raw files, and

.cmd derived from .inp file; contains only the command part of the input line.

The MRP tool uses the .cmd file to scan for MRPs, while keeping track of where instances of an MRP occur in both the .inp and .raw files (i.e., the line numbers in these files at which the instances occur). The relationship among these files is shown in Figure 2.

As a result of the scan, the tool produces a list of MRPs found in the transcript (see Figure 3). By definition, each of these MRPs has at least two instances that occur at distinct positions in the transcript file. An interface evaluator analyzes the transcript by examining MRPs in the list and

selecting those that are "interesting." For example, an evaluator might notice that a certain MRP has a large number of instances or that an MRP is a sequence of repetitions of the same command. For each MRP that is "interesting," the evaluator may examine the instances of that MRP by viewing the complete input lines which make up the instance and, if more detail is needed, by examining the output lines associated with those input lines. Any information discovered can be copied to another file.

In addition, the evaluator can select MRPs from the list based upon criteria such as length of an MRP. A new list is created containing all MRPs which satisfy the selection criteria. This "filter" list can be refined by further applications of selection criteria.

A command occurrence table is generated as a by-product of the MRP detection operation. The evaluator uses a separate program to compute statistics such as usage frequency, number of sessions represented by the transcript and maximum, average, and minimum number of commands executed per session.

5. TESTING MRP USEFULNESS

The usefulness of MRPs in interface evaluation was studied by analyzing the interface of GIPSY, a large and complex system in regular use at the Spatial Data Analysis Laboratory at Virginia Tech. GIPSY was selected as the testbed because of this fact and because its users were known to complain about it being hard to use. Thus if MRP analysis did not reveal any problems with the interface, it was unlikely that MRP analysis would prove useful in interface evaluation. If MRPs did point to problems with the interface, that would indicate the method was promising, and that further research should be undertaken.

5.1 GIPSY

GIPSY is an image processing system designed to run on the VAX [TM] series of computers [9]. At present, it supports over three hundred and fifty image processing algorithms, ranging from the classical to the most advanced. It is in use at numerous sites throughout the United States. Its interface is a highly modal command line interpreter which supports execution of local operating system commands from within GIPSY.

5.2 Procedure

GIPSY was modified to record all user keystrokes and system output on a user session basis. All keyboard and screen activity was recorded since it was not known precisely what information would prove useful. The data were collected over three months.

The evaluation procedure is illustrated in Figure 4. First, the collection period transcripts of each user were concatenated in chronological order, producing a single raw transcript representing all the sessions of that single

[TM] VAX is a trademark of Digital Equipment Corporation.

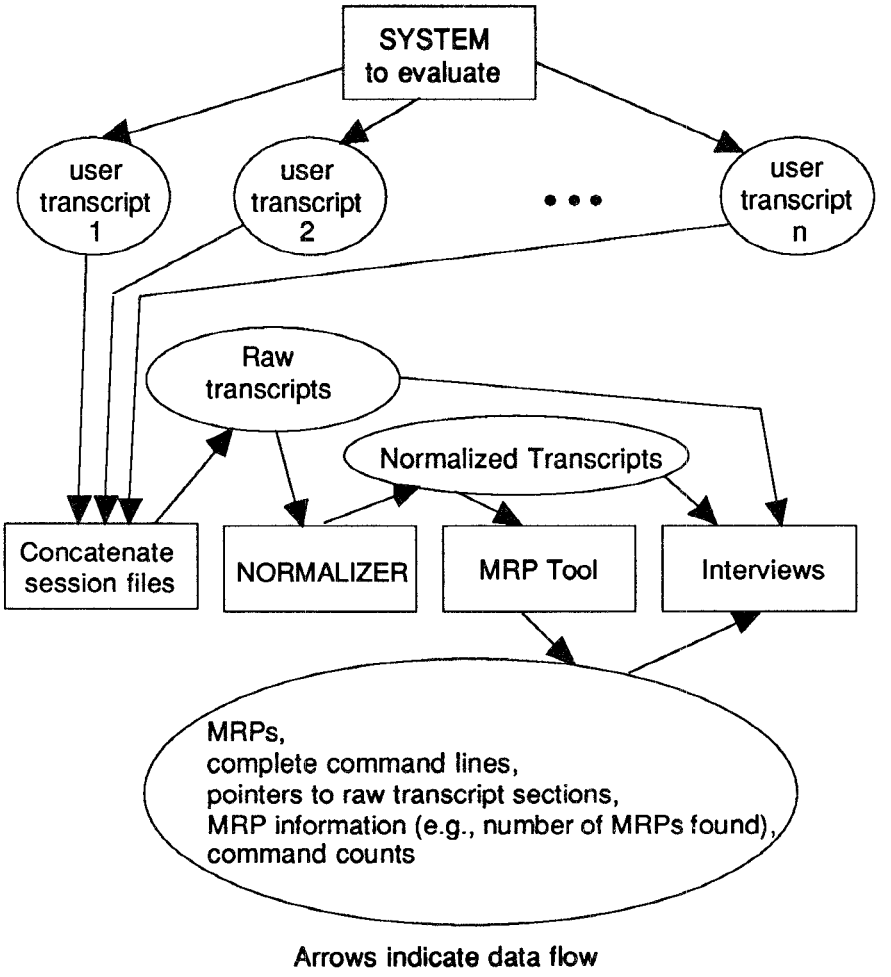

Fig. 4. MRP analysis tools and the evaluation process.

user. This reduced the number of files to be analyzed and allowed a single analysis across all sessions of that user. After the raw transcripts were translated into a standard format, MRPs were extracted. The MRP list was scanned by the evaluator, and where necessary, complete command lines corresponding to the MRPs were reviewed and the corresponding raw transcript sections examined. Two GIPSY users were interviewed subsequently. This step was essentially a debriefing structured by the MRP analysis results. The findings were then summarized.

6. RESULTS

The transcripts of 17 users totalling 300,000 lines of raw transcripts were normalized, yielding 17,086 command lines. From these command lines, 3,523 MRPs were detected, resulting in the discovery of 12 problem classes in

Table I. Problem Classes Detected Using MRPs

PROBLEM CLASS	PREVIOUSLY KNOWN
Consecutive Invocations	
GIPSY Response Time	yes
User Macros	
Poor Feedback	yes
SYSTEM Response Time	
Error Messages	yes

PROBLEM CLASS	PREVIOUSLY KNOWN
Parser	
EZPLOT	
PRTPF	
LWPIC	
EXPL	yes
Runfiles	

Table II. Some Information from MRP Analysis of Seventeen Transcripts

USER	NUMBER OF LINES	NUMBER OF MRPS	LENGTH OF LONGEST MRP	TYPE ONE MRP PRESENT?	TYPE TWO MRP PRESENT?	MRPS PER COMMAND LINE
U01	118	26	8	0	1	.22
U02	85	13	17	1	0	.15
U03	2728	520	26	1	0	.19
U04	76	10	9	0	1	.13
U05	50	13	4	1	0	.26
U06	223	34	6	1	1	.15
U07	220	41	6	1	0	.19
U08	4778	951	29	1	1	.20
U09	66	9	7	1	0	.14
U10	234	44	6	1	0	.19
U11	2867	681	21	1	1	.24
U12	22	3	6	1	0	.14
U13	121	14	26	0	0	.12
U14	1079	268	25	1	1	.25
U15	2248	476	19	1	1	.21
U16	1262	220	37	1	1	.17
U17	909	200	17	1	0	.22
Totals	17086	3523	269	14	8	average =.21

Average number of MRPs per user = 207
Average length of longest MRP = 16
82% of users had type one MRP
47% of users had type two MRP

GIPSY, four of which were previously known (see Tables I and VI). Analysis based solely on MRP lists revealed several interesting and specific problems with GIPSY, yet left several questions. Interviews validated certain deductions made in the first analysis and answered questions of that stage.

The next section describes the first four problems listed in Table I and the succeeding section deals with the remainder.

6.1 MRP List Analysis

MRPs were analyzed over an eight-hour period, the bulk of which was spent studying the MRP lists, each of which took a few seconds on average to generate. Table II gives, for each user, the number of MRPs detected, the

```
 0)  DSPLY
 1)  DSPLY
 2)  DSPLY
 3)  DSPLY
 4)  DSPLY
 5)  DSPLY
 6)  DSPLY
at:   656   657   658   1137
      1138  1139  1565  1574...
Total number of positions =   8.
```

Fig. 5. An instance of a type one MRP: consecutive invocations of the same command.

Table III. Commands Found in Type One MRPs (Definitions from [9])

BINSIF	Convert binary data to Standard Image File (SIF) format
CHRSIF	Convert a character file to a SIF file
DSPLY	Display an image in the SIF format
EXPL	Explain GIPSY commands
INVIMG	Invert the gray levels of an image
MHIST	Computes histogram of an integer image
RUN	Switch command input from terminal to RUN file
SBIMG	Extract a subimage
SIFCHR	Convert a SIF image into a character file
SYSTEM	Run local system command

length of the longest MRP, and whether or not two types of MRPs were found.

Consecutive invocations. Given the large number of detected MRPs, only those which violated expected usage patterns of commands in general were pursued. For example, type one MRPs are those consisting of consecutive invocations of the same command (see Figure 5). Ten of the 350 GIPSY commands produced this type of MRP (see Table III). Type one MRPs may indicate that a user needs to perform the same command on several objects, that a user is searching for a way to do something or that the user is "fine-tuning" a single object. For example, a user may have a list of files that need to be converted from one format to another, a user may be repeating the same command with different arguments to discover the proper syntax or a user may be debugging a macro. In the first case, a possible remedy could be to allow an arbitrary number of arguments for each such command. The second case would suggest a review of the syntax of that command, while the last case requires a closer study of the nature of the "fine-tuning." The fact that 82% of the sample users exhibit this MRP type suggest that this indicates a problem inherent in the interface design, rather than a collection of user idiosyncrasies.

GIPSY response time. Type two MRPs consist of consecutive lines where no commands were entered (see Figure 6). These indicate anomalous use of

```
0)
1)
2)
3)
4)
5)
6)
at:    511    667    668    669
670    671    672    673...
Total number of positions =   21.
```

Fig. 6. An instance of a type two MRP: empty command lines.

Table IV. Sample MRPs

A	B	C	D	E	F
235 times: CHRSIF A > B EXSIF B DSPLY B	36 times: BINSIF A > A DSPLY A	26 times: CHRSIF A > B PLTSIF B > C PRINT C	19 times: SOBEL A > B DSPLY B	13 times: TRSLD A > B PLTSYM B > C	29 times: STOP STOP

the command line terminator, which for GIPSY is the carriage return. This MRP type may be due to factors such as poor keyboard design, defective keyboards or long response times. However, the high proportion of users who exhibit this MRP and the long experience designers have with keyboards suggest that GIPSY response time is the more likely cause.

User macros. Table IV shows six sample MRPs. The notation "X A > B" means the user invoked command "X" using file "A" as input and file "B" as output. Notice that most of these MRPs use the output of one command as input to a subsequent command. Such MRPs suggest the development of specific macros and even identify the parameters and local variables of the macro, in addition to the sequence of commands that make up its body.

MRP "A" in this table indicates that a user tends to modify (using EXSIF) a file that was just converted with CHRSIF. The DSPLY command is used to verify the modifications. Because this MRP occurs frequently, a macro which combines these commands may be desirable.

Poor feedback. MRPs "A" through "E" are indicative of loss of state information. These show the user confirming the effects of an image manipulation command. They indicate the strong need for feedback in the interface (a previously known problem). Instead of typing an output command each time (e.g., DSPLY), the image manipulation commands could have an option to redisplay the image after processing it.

6.2 Structured Interview

After studying the MRP lists, it became clear that some MRPs could be explained only by asking users what they had been doing at the time. Two of the seventeen users, U17 and U14, were selected to be interviewed

approximately one year after the data were collected. Both users were GIPSY experts and did GIPSY development work. The basis for the selection was availability, since most of the users could no longer be contacted (e.g., students in the image processing class).

In each interview, the user was presented the list of MRPs detected in that user's transcript and asked to remember what he had been doing or trying to do. When viewing MRPs alone, neither user could remember what he had been doing, but when the MRP tool was used to show the complete command lines corresponding to the MRPs, they were able to remember some tasks. When shown sections of raw transcript corresponding to the MRPs, both users remembered almost all tasks. This is a remarkable result, considering that one year had elapsed since the data were collected.

SYSTEM response time. One MRP which required clarification by interview was "F" (see Table IV). MRP "F" is highly unusual because it shows that in several sessions users did not invoke any GIPSY commands, that is, users would run GIPSY then quit immediately. A reasonable inference is that the user forgot to do something before invoking GIPSY. However, GIPSY provides a command (SYSTEM) which allows users to access the operating system from within GIPSY. It was thus curious why users would quit GIPSY when they could have used the SYSTEM command. Probing both users on this point revealed that they perceived the response time for the SYSTEM command to be excessive and had developed a decision rule to the effect that, for some commands, it is faster to leave GIPSY, invoke the operating system commands and then return to GIPSY, rather than to use the GIPSY SYSTEM command.

Error messages and the parser. MRP "F" prompted both users to make further specific comments about the interface. For example, both users said that another reason for not using the SYSTEM command was that the error messages it returns are not as clear as those returned by the operating system. U14 started to recall several other problems with the SYSTEM command, one of which involved the GIPSY parser interpreting a SYSTEM command as a GIPSY command.

EZPLOT. Table V lists three anomalous MRPs. MRP "G" shows the user alternating between two commands, EDGEX2 and EZPLOT, but with the curious characteristic that each command was a single session. When questioned, U17 stated that he used many batch files. Furthermore, he stated that EZPLOT did not have good support for multiple plots. This MRP shows compensatory behavior—the use of batch files to perform multiple plots.

PRTPF. In MRP "H," %P is a command that allows the user to repeat a previous command. In this case the raw transcript indicated that U14 was using PRTPF to test for the length of a file. U14 confirmed this inference and stated that the command that provides information about property files did not show how long a file was.

Table V. Anomalous MRPs

G	H	I
STOP	RUN REORDER.RUN	RUN OVER.RUN
EDGEX2	PRTPF	$
STOP	%P	RUN OVER.RUN
EZPLOT	%P	$
STOP	%P	RUN OVER.RUN
EDGEX2	%P	$
STOP	%P	RUN OVER.RUN
EZPLOT	%P	$
STOP	%P	RUN OVER.RUN
EDGEX2	%P	$
STOP	%P	RUN OVER.RUN
EZPLOT		$
STOP		RUN OVER.RUN
EDGEX2		$
STOP		
EZPLOT		
STOP		

LWPIC. Another interesting MRP from U14 was

```
ARITHM  A > B
EXSIF   B
LWPIC   B > C
```

which indicates inadequate application functionality. This MRP occurs eight times in U14's transcript. U14 is apparently modifying file A before printing it out. Debriefing U14 confirmed this: U14 stated that LWPIC could not print low-contrast image files, so U14 first had to increase an image file's contrast using ARITHM and EXSIF.

Note that although this MRP did identify a macro, in this case the macro constituted *compensatory* behavior—the real problem was LWPIC. This result shows that MRPs identify repeating usage patterns, but not the *reasons* for the repetition. Examination of the contexts in which the MRP occurs is required to deduce those reasons.

EXPL. U14 was also asked about the type 1 MRP where EXPL was the repeated command. This was considered an anomaly as U14 was an expert user. U14 replied that other users would ask him questions about GIPSY and he used EXPL to help him answer those questions. This is an interesting result because it confirms an observed tendency of naive users to ask other users questions rather than use the help system. Thus this MRP indicates a need for further investigation of the help system, in particular the entries for which U14 was consulted. Note that although it was previously known that help entries were poorly written, MRPs identified specific help entries to study.

Table VI. Previously Known GIPSY Problems

DETECTED USING MRPS	NOT DETECTED
help entries are not written well	difficult to find and determine which command to use
error messages not meaningful to user	command names are poorly abbreviated
GIPSY response time can be slow	command guesser often guesses wrong command
poor feedback of image manipulation commands	slow input methods

Runfiles. Another problem both users complained about as a result of reviewing the MRPs and transcripts was that runfiles, (i.e., GIPSY macros) caused GIPSY to terminate when the runfile terminated. U17 explained that MRP "I" was an attempt to fix this problem by alternately modifying then running the runfile. U17 did not succeed in his attempts. This problem is an example of users trying to circumvent an application malfunction.

6.3 Known GIPSY Problems

An informal evaluation of GIPSY was performed prior to MRP analysis. This involved using GIPSY, reviewing manuals, and talking with users and the GIPSY site manager. The problem classes discovered are listed in Table VI.

Although only four of the previously known problems were detected, MRP analysis discovered eight unknown problems. This can be explained in part because the newly discovered problems are very specific, dealing with particular commands such as LWPIC. Such problems are usually discovered only after prolonged use, hence would not be found during a relatively brief evaluation of the system. *GIPSY users were notorious for their lack of response to requests from developers for comments and problems encountered. Therefore specific problems were not likely to be known to the developers.*

6.4 Problems With the Technique

It is interesting to note that relatively few GIPSY problems were detected. There are two reasons for this result. First, GIPSY users tended to use relatively few commands, ranging from four to sixty commands out of 350 available. Because there were no usage data for unused commands it was not possible to detect errors associated with those commands. (However, the fact that only a few commands are ever used can lead to a redesign of the command set.) Second, a large number of MRPs were detected. This forced the evaluator eventually to examine only those MRPs which occurred more than the average number of times.

Since the number of detected MRPs reach into the hundreds for almost half of the transcript files (see Table II), the problem of tedium is resurrected. Figure 7, however, shows that the relationship between number of command lines in a transcript and number of detected MRPs is linear. This means that part of the reason why numerous MRPs were detected is because there was so much information initially.

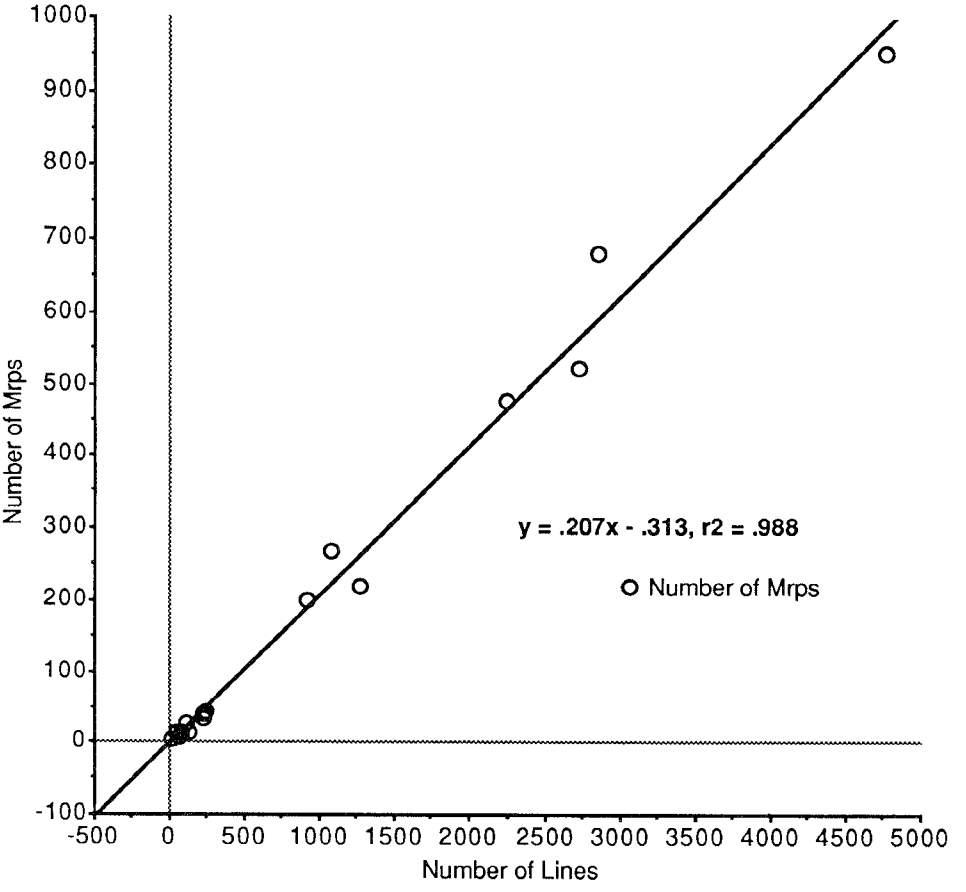

Fig. 7. Plot of number of lines versus number of MRPs detected

A related problem is the large amount of noise present. Some MRPs did not appear to indicate anything interesting about the interface, while other MRPs were actually substrings of larger MRPs and repeated information provided by the longer MRP. The first case involved MRPs which occurred at only two positions, thus this type of noise may be attributable to chance. Some sort of statistical filtering of MRPs, such as reporting only those occurring at greater than chance levels, may reduce this kind of noise. The second type of noise is due to the definition of an MRP and is probably the main contributing factor to the large number of MRPs detected, considering the vastly greater number of shorter MRPs (See Figure 8).

7. CONCLUSIONS

The results demonstrate that the technique was useful for finding specific problems in the GIPSY interface (e.g., the problems with the LWPIC command). The MRP algorithm found repeating patterns of user actions in

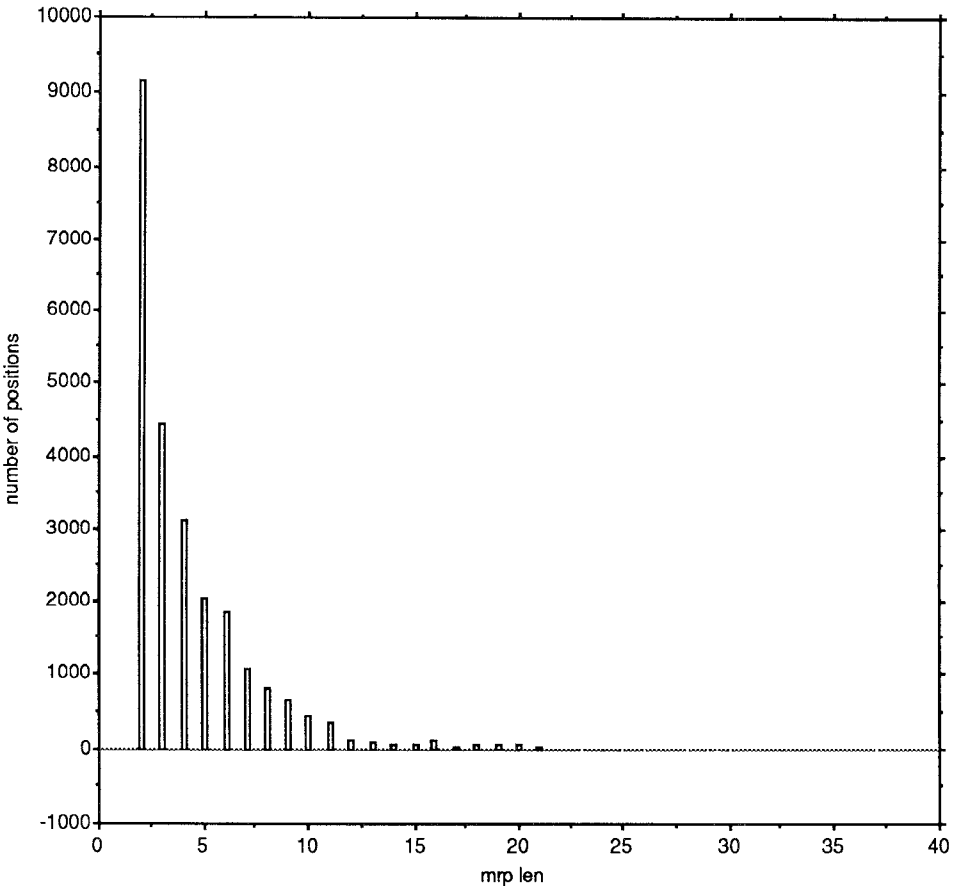

Fig. 8. Distribution of number of positions an MRP occurs at according to its length.

the transcripts and these patterns indicated aspects of the GIPSY user interface which needed attention. Although MRPs do not show a problem's cause, MRPs did identify specific, real problems.

7.1 Advantages

The distinguishing advantage of this technique is that by making no assumptions about what tasks are important to users, it reveals patterns of user actions which can indicate interface problems. Consider the user macros listed in Table IV. It is difficult to imagine how searching for fixed patterns of user actions would uncover such unknown patterns. The specific GIPSY help entries which needed further investigation would be difficult to identify by using summary data such as command frequencies. Scanning for repetitions revealed these specific patterns.

The technique was also useful in preparing and conducting the structured interviews. Users can be shown MRPs and *recognize* problems they

experienced rather than be forced to *recall* problems when confronted with the standard question, "What did you dislike about the system?" This was particularly useful in GIPSY since attempts by the developers to get users to report problems failed. In fact, a GRIPE command which allowed users to make problem reports while in GIPSY was used only *once* since it was installed.

Another benefit is the speed of the tool both for analyses and interviews. Long processing times would tend to discourage evaluators from using the technique, while rapid processing makes possible the analysis of large amounts of data. Quick access to information such as MRPs, complete command lines and raw data is important in interviews because a busy user has graciously volunteered some time.

Finally, the specificity of the problems identified by this technique is useful to developers responsible for maintaining the system. The ability to point to and present raw transcript sections corresponding to problems is a tremendous advantage in debugging and is not usually provided by user complaints.

7.2 Limitations

The technique's central limitation is the type of information it provides. MRPs focus on specific, detailed problems encountered by users. General aspects of the usability of the system are not directly exposed. For example, although this technique identified the previously known GIPSY problems of poor response time and error messages, it did not show the well-known deficiency of GIPSY, which is the great difficulty users have in discovering which command to use for the task they wish to perform. However, the mere fact that GIPSY has more than three hundred and fifty commands should immediately raise concerns about the accessibility of those commands. This type of information is readily discernible from even a short exposure to GIPSY, or casual conversation with users. It is the details of everyday use that are missed in such dialogue, probably because users have adapted to those problems and thus do not talk about them. It is precisely those details which the MRP technique addresses and has been shown to detect.

One might argue that since users have adapted to those problems, it would not have been cost-effective to fix them. This statement has some validity, yet it ignores the fact that such adaptation has associated costs in terms of increased performance times and lower user satisfaction. Each adaptation is a set of tasks the user has to perform either to avoid some undesirable interface behavior or to effect some missing functionality. The extra time involved in such tasks cannot be denied.

A more detailed limitation is that studying MRPs alone does not produce as much insight as studying complete command lines. This is because MRPs show only command names, while complete command lines show arguments as well. Similarly, interviews generated more information. In general, the technique does not use a lot of other information that could be part of a transcript file. For example, error patterns, help usage, user think and performance times, and system response times could all be recorded on the transcript. In fact, some MRPs showed classes of usage patterns such as repeated invocations of a command on an object (type one MRPs) or feeding

the output of one command into the input of another (user macros). A broader evaluation tool should encompass these diverse elements.

Another limitation is that the analysis depends heavily upon the evaluator's skill and knowledge of both the evaluation method and the application system being evaluated. Human judgement is required in deciding which MRPs to examine further, in making deductions and in proposing changes. Also, although a sequence of commands may be repeated often, the conclusion is not necessarily that a macro is needed. The repetition may be a user adaptation to a different interface problem. Consequently, *the MRP tool is more analogous to a microscope than to a weighing scale*: the tool provides an ability to analyze transcripts at different levels of detail, rather than a measure of some interface characteristic.

Finally, the technique was applied to a system with a command line interface, ignoring the important class of direct manipulation interfaces. This was a deliberate decision to simplify the problem, especially since the viability of the technique was the object of investigation.

7.3 Ethical and Legal Considerations

This evaluation technique is based on the collection of user inputs and outputs during the ordinary use of a system. This can raise ethical and legal problems with regard to privacy issues—a system which is instrumented to collect such data could be construed as a form of wire-tapping.

As was done for this study, users should be made aware of the collection of their keystrokes and screen displays and of the purpose of the collection. In addition, users should be given the opportunity to specify that they do not wish their data to be collected. Moreover, when users have given permission for their data to be collected, access to that data must be restricted only to authorized persons and not released without prior user consent.

This disclosure of data collection may result in few or no users willing to participate in the evaluation activity. For certain systems such as electronic mail, this situation is almost to be expected. Even for other seemingly innocuous systems, users may view the data collection as work/performance monitoring and for this reason decline to participate. Despite the possibility of having no users willing to participate, evaluators have an ethical responsibility to disclose the data collection.

7.4 Recommendation

From the previous discussion it can be seen that MRP analysis is good for working at the detailed level of interfaces, but not at a general level. At this point, MRP analysis would be most useful in the summative evaluation, beta-test and maintenance phases of software development, especially when used in conjunction with user interviews.

8. FUTURE WORK

8.1 Reducing Analysis Time

Considering the graph of Figure 7, a large scale study would inundate the evaluator with MRPs. One solution would be to limit the amount of data collected. The maximum amount to be collected can be estimated by

determining how many MRPs the evaluator can analyze in the time scheduled and solving an equation similar to that shown in Figure 7 for the number of command lines to collect.

Another way is to develop filtering algorithms or heuristics which can be applied to the MRP list. A simple attempt at heuristics was implemented in the MRP tool, where the evaluator examined only those MRPs whose lengths were greater than average. Another heuristic might be to examine only MRPs containing the most frequently used commands. Algorithms that reduce the amount of redundant information and a reevaluation of the MRP definition (specifically removing the independent occurrence condition) should be investigated.

8.2 Similarity Indicator for Command Lines

It was noted that complete command lines (MRP instances) were necessary for analysis. A means of representing those portions of command lines which differed across instances of an MRP might reduce the need to examine complete command lines. A similarity representation might show, for example,

```
EXSIF f??.dat > f??.out
EZPLOT f??.out
at positions: 1, 234
```

where question marks indicate differences in the MRP instances. This representation technique could be implemented using a string to string correction algorithm, similar to that used by the *diff* program in UNIX.

8.3 Use as a Support Technique

Another interesting avenue to explore would be MRPs as an aid to videotape-based interface analysis. A major problem with videotape is having to review the tape manually for critical incidents. Since the MRP technique points to potential problem sections in the transcript, it could also be used to indicate similar sections of videotape. This may be achieved by inserting video frame numbers into the transcript file.

8.4 Time Stamping

MRPs might provide better information if combined with time-stamping data. This would enable a quantitative expression of user effort represented by each MRP, and could provide evaluators another means of selecting MRPs on which to focus: those with elapsed times greater than expected.

8.5 A Transcript Analyzer

Because of the central limitation of MRP analysis of transcripts, a broader transcript analyzer should be considered. This analyzer would provide the evaluator with a suite of analysis tools, of which the MRP tool is but one. Other tools that would be useful include pattern matching tools, grammatical analysis tools and statistical tools. These tools may act as a preprocessor for

transcript data and still other tools may locate or count complicated relational events of the form (A preceded by B and two consecutive occurrences of A). Such tools would permit the MRP analyzer to focus on particular aspects of a transcript, and they would allow an investigator to play with the data, perhaps to test whether certain types of behavior suggested by MRP analysis are in fact occurring elsewhere in the dialogue.

Earlier we described the possible use of AWK, LEX and YACC in constructing the normalizer. AWK is effective in locating complicated patterns in a transcript, whereas LEX and YACC are useful tools for building procedures that identify complicated relationships among patterns. These are the types of tools that are needed to augment the MRP toolkit; because of their sophistication they need powerful human computer interfaces supported by a UIMS.

9. SUMMARY

This paper reported on the development of a new technique for evaluating interfaces by analyzing user session transcripts. The technique involved the detection of repeated user actions in those transcripts and is based on the hypothesis that repetition of user actions is an indicator of potential interface problems. The concept of maximal repeating patterns, or MRPs, was developed as a means of defining the repetition.

A tool was developed which extracts MRPs from transcripts in $O(n^2)$ time. This tool also enabled the evaluator to manage lists of MRPs, select MRPs based on their length or frequency and view the raw transcripts pointed to by those MRPs.

The technique was tested on GIPSY, an image processing system in use at several sites throughout the country. The data were collected from actual users at one site over three months and the analysis involved both an independent study of the MRP lists and structured interviews of two users. The technique was shown to provide useful information about the GIPSY interface by revealing several specific problems.

Advantages of the technique include detection of unknown patterns which potentially show problems users were having, usefulness in preparing structured interviews and speed and ability to scan large amounts of data. The technique's limitation is that the information it provides is at a detailed level and does not directly indicate general problems. This implementation also produces MRPs which are redundant or seem insignificant. Despite its limitations, the technique provided useful information about GIPSY's interface. In addition, it is important to remember that MRPs represent the experience of users in their natural work context. As such, MRPs are a source for the discovery of how users actually use the system.

ACKNOWLEDGMENTS

We would like to thank the SDA Lab at Virginia Tech for serving as a testbed for the research and would like to express our appreciation to the reviewers for their helpful comments.

REFERENCES

1. AHO, A., WEINBERGER, P., AND KERNIGHAN, B. AWK—A pattern scanning and processing language. *Soft. Prac. and Experience* (July 1978).
2. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms*. Harrison Ed. Addison-Wesley, Reading, Mass., 1974.
3. BLUMER, A. ET AL. The smallest automaton recognizing the subwords of a text. *Theor. Comput. Sci. 40* (1985), 31-35.
4. CARD, S. K., MORAN, T. P., AND NEWELL, A. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum, Norwood, N.J., 1983.
5. CARROLL, J. M., AND ROSSON, M. B. Usability specification as a tool in iterative development. In *Advances in Human Computer Interaction*. Hartson Ed. Ablex, Norwood, N.J., 1985.
6. COHILL, A. M., AND EHRICH, R. W. Automated tools for the study of human/computer interaction. In *Proceedings of Human Factors Society 27th Annual Meeting* (Norfolk, Va., Oct. 10-14). Human Factors Society, 1983, pp 897-900.
7. EASON, K. D. Towards the experimental study of usability. *Behav. Inf. Tech. 3*, 2 (1984), 133-143.
8. EHRICH, R. W. *The DMS Multiprocess Execution Environment*. CSIE-82-6, Virginia Tech Computer Science Dept., 1982.
9. GARLAND, E., AND EHRICH, R. W. *A GIPSY Primer*. Spatial Data Analysis Laboratory, Blacksburg, Va., 1987.
10. GOOD, M. The use of logging data in the design of a new text editor. In *Proceedings of CHI'85, Conference on Human Factors in Computing Systems*, (San Francisco, Apr. 14-18, 1985). ACM, New York, 1985, pp. 93-97.
11. GOULD, J. D., AND LEWIS, C. Designing for usability: Key principles and what designers think. *Commun. ACM. 28*, 3 (Mar. 1985), 300-311
12. HANSON, S. J., KRAUT, R. E., AND FARBER, J. M. Interface design and multivariate analysis of UNIX command use. *ACM Trans. Off. Inf. Syst. 2*, 1 (1984), 42-57.
13. HARTSON, H. R., AND HIX, D. Toward empirically derived methodologies and tools for human-computer interface development. *IJMMS. 31* (1989), 477-494.
14. JOHNSON, S. C. *YACC: Yet Another Compiler Compiler*. Computing Science Tech Rep 32, Bell Laboratories, Murray Hill, N.J., 1975.
15. KARP, R. M., MILLER, R. E., AND ROSENBERG, A. L. Rapid identification of repeated patterns in strings, trees, and arrays. In *Proceedings of Fourth Symposium on Theory of Computing* (1972), pp. 125-136.
16. LESK, M. E. *Lex – A lexical analyzer generator*. Computing Science Tech. Rep. No. 39, Bell Laboratories, Murray Hill, N.J., 1975
17. MACKAY, W. E., GUINDON, R , MANTEI, M., SUCHMAN, L., AND TATAR, D. G Video: Data for studying human-computer interaction In *Proceedings of CHI'88 Conference on Human Factors in Computing Systems* (Washington, D.C., May 15-19, 1988). ACM, New York, 1988, pp. 133-137.
18. NEAL, A. S., AND SIMONS, R. M. Playback: A method for evaluating the usability of software and its documentation. In *Proceedings of CHI'83 Conference on Human Factors in Computing Systems* (Boston, Mass., Dec. 12-15, 1983). North-Holland, Amsterdam, 1983, pp. 78-82
19 OLSEN, D. R., AND HALVERSEN, B. W. Interface usage measurements in a user interface management system. In *Proceedings of ACM SIGGRAPH Symposium on User Interface Software* (Banff, Alberta, Canada, Oct 17-19, 1988). ACM Press, New York, 1988, pp. 102-108.
20 SAS INSTITUTE. *SAS User's Guide*. Helwig and Council, Ed. SAS Institute, Raleigh, N.C , 1979.
21. SHNEIDERMAN, B Direct manipulation: A step beyond programming languages. *IEEE Comput. 16*, 8 (Aug. 1983), 57-69.
22. SIOCHI, A. C. Computer-based user interface evaluation by analysis of repeating usage

patterns in transcripts of user sessions. Dissertation. Virginia Polytechnic Institute & State Univ., Blacksburg, Va., 1989.

23. SIOCHI, A. C., AND HARTSON, H. R. Task-oriented representation of asynchronous user interfaces. In *Proceedings of CHI'89 Conference on Human Factors in Computing Systems* (Austin, Texas, April 30–May 4, 1989). ACM, New York, 1989, pp. 183–188.
24. WEINER, P. Linear pattern matching algorithms. In *Proceedings of IEEE 14th Annual Symposium on Switching and Automata Theory*, 1973, pp. 1–11.
25. WHITESIDE, J., BENNETT, J., AND HOLTZBLATT, K. *Usability engineering: Our experience and evolution*. DEC-TR 547, Digital, 1987. To appear as a chapter in *Handbook of Human-Computer Interaction*, M. Helander Ed., North-Holland, Amsterdam.
26. WILLIGES, R. C. The use of models in human-computer interface design. *Ergonomics 30*, 3 (1987), 491–502.
27. WILLIGES, R. C., WILLIGES, B. H., AND ELKERTON, J. Software interface design. In the *Handbook of Human Factors*. Salvendy, Ed. Wiley, New York, 1987.