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

Antonio Siochi
Roger W. Ehrich

TR 90-15

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

Antonio C. Siochi
Roger W. Ehrich

Computer Science Department
Virginia Tech
Blacksburg, VA 24061-0106

ABSTRACT

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 non-invasively 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 sketches the detection algorithm and 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.

KEYWORDS & PHRASES: user interface evaluation, transcript analysis, repeated usage patterns, usability, maximal repeating patterns, user interface management systems

INTRODUCTION

The past decade has seen the emergence of a new consciousness in the interactive software community. Users are demanding that their software not only provide computational power, but also that the software be *usable*. Managers are growing aware of the long-term costs associated with poor usability, and developers are beginning to design systems that *fit the user* as well as meet functional requirements.

This push for usability has forced the modification of standard software development methodologies. There is growing support for the process of iterative design and rapid prototyping—a design process in which user feedback is an essential element. In support of this, Carroll & Rosson [4] argue that design is a dynamic process that does not fit into a strict top-down or hierarchical decomposition scheme. They stress the importance of an iterative design process, supported by empirical evaluations of system prototypes:

> The most important aspect of the empirical approach is that it encourages the discovery of design solutions which on purely analytic grounds might have been missed. ... Our view is that design activity is essentially empirical. This is not because we "don't know enough yet," but because in a design domain we can never know enough.

A number of authors have argued in favor of an iterative design methodology [9, 11, 22, 23]. Whiteside, Bennett, & Holtzblatt [21] state several reasons for using an iterative design methodology. First is that "User testing is the most reliable way to debug the user interface." Second, the feedback provided by frequent testing increases the amount of control managers have over the development project. Last, because some version of the system must be available for testing, developers can respond more quickly to management decisions which might advance the ship date.

Against this background, the need for evaluating the human-computer system, especially its user interface, stands sharp and clear. In response, many methods and techniques for evaluating user interfaces have already been developed. Each has its strengths, limitations and costs. The contribution of this research is a new, relatively low-cost evaluation technique 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 which contain 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.

EMPIRICAL USER INTERFACE EVALUATION

If one accepts the arguments in favor of an iterative development methodology, then one is faced with the task of empirical evaluation and the problem of locating interface weaknesses as quickly and inexpensively as possible. Therefore, a brief review of existing empirical methodologies will help set the context for the new methodology involving maximal repeating patterns.

Formal Experiments

Formal techniques employ controlled experiments and statistical methods to measure specific aspects of a human-computer interface. Experiments are usually performed in a laboratory where lighting, temperature, and ambient noise can be held constant from subject to subject. Before the session starts, the subject often completes a questionnaire which seeks to classify the subject in relation to the other subjects or to some user population. The experiment is often videotaped, and the subject may be assigned a standard set of tasks to complete. During the session, the experimenter is required to provide each subject with the same information and must be careful not to ask leading questions. At the end of the session, the user may be asked to answer an exit questionnaire. The data are later analyzed for verbal protocols, critical incidents, time to accomplish the benchmark tasks, number of errors, and other similar measures. Analysis typically involves reviewing the videotape of the session, and therefore takes at least as long as the session itself. In practice, however, analysis takes many more times that amount. Indeed, Mackay [14] has stated that for a one hour session, the analysis can take a day or more to perform.

Contextual Research

Apart from long analysis times, Whiteside, Bennett, & Holtzblatt [21] state that a problem with experimental methods performed in a laboratory is the lack of a natural work context. For example, benchmark tasks reflect a third party's judgement about what is important to the user, "... the deliberate restricting of focus..., has the effect of making an a priori value judgement that these operations are at the heart of usability for the system." They advocate a *contextual research* method, a method of collecting data about the user's actual, everyday experience of using the system.

One way of collecting such data is the *contextual interview*. This consists ideally of visiting users at their place of work, videotaping, and interviewing them as they work. The purpose of the interview is to capture the user's experience of using the system at the moment of that experience. The observer can immediately validate any interpretations by asking the user. In effect, data collection and analysis are performed concurrently. Unfortunately, this technique examines a thin slice of the user's daily experience with the system. It is also quite expensive to send a contextual research team to each site. An extended visit, although it would certainly reveal a lot more about the customer, would cost correspondingly more.

Transcript Analysis

Each of these methods has its own strengths and weaknesses. 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. Contextual research techniques can provide results in less time and with less money than formal experiments; however they are ad hoc, and the results can not be validated in the strict experimental sense. Both the formal experimental techniques and contextual research techniques are invasive as well, although in different ways: the experimental techniques pluck users out of their natural environments, whereas the contextual techniques insert the researcher into that environment.

One empirical technique that preserves 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.

The advantages of transcript analysis are numerous. First, the data represent the user's interaction with the system under actual working conditions, which satisfies Whiteside, Wixon, and Holtzblatt's concern. Second, since the data are stored in on-line files, they are accessible to algorithmic analysis and data reduction techniques. Such analytic tools liberate the evaluator from 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 certain analyses involving vast quantities of data, which would never have been considered with videotape.

However, there is still the matter of collecting the data. As will be seen in the examples that follow, the collection techniques range from ad hoc modification of the interface to using external hardware to collect keystrokes. Ideally, transcript analysis should be supported by a User Interface Management System (UIMS). This would eliminate the need to modify the interface to collect user input. It would also provide integrated support for the data collection, management and analysis activities of an evaluator. This has been argued for by Ehrich [6], Olsen & Halversen [16], and Hartson & Hix [11]. The major benefit of this integration is that it gives explicit recognition to the essential nature of evaluation in the iterative development methodology.

Certainly this technique does not capture as much data as videotape does. For that matter, neither videotape nor this technique can capture user *intent*. 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 the hours of videotape with megabytes of files.

There are several examples of this technique in the literature. Cohill & Ehrich [5] describe a set of programs and routines developed to collect keystrokes and state information, and compress the raw data. Because of limited resources, they could not afford the usual data collection procedures. Their budget allowed for only one experimenter, and the schedule required running two to four subjects at a time. Since the variables they were measuring (time spent in help, number of times help was invoked, frequency of command use) did not require human judgement, an automated technique was used. They inserted calls to the metering routines at strategic points in the code of the system they were investigating, and then using their tools, reduced the resulting data to a form suitable for immediate input to SAS. 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.

As part of the special services of DMS, an early UIMS, Ehrich [6] provided similar logging routines that tool builders could use in order to provide interface developers with logging services. Olsen & Halversen [16] also implemented this concept and studied the issues involved in such an enterprise. Their metrics included 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 is also used by the metrics computation and report generation program to generate human-readable reports. The reports present, for each metric, a list of commands ranked from worst to best for that metric. This report is then used to detect problems in the interface.

Neal & Simons [15] utilize a different approach. They set up one computer to intercept, record, and time-stamp each keystroke. The keystroke was then passed to the other computer which ran the application system. An advantage of this method is that no modifications to the application system are 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 & Simons found their methodology to be "... very effective for objective evaluation and comparison of software including the user interface design and software documentation."

Good [8] analyzed existing logging 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, e.g., the inverted-T layout of the cursor keys. The 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 the design decisions taken and to provide feedback for the next design iteration.

Hanson, Kraut, & Farber [10] collected command usage data on UNIX™ 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 modularity of each command, i.e., the degree to which a command is used together with many other commands. Treating this same matrix as a contingency matrix enabled Hanson, Kraut, & Farber to determine the sequential dependencies among commands. They prescribed some restructuring of the UNIX interface based upon these results.

The examples presented above have a common set of analyses. These revolve around the 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 collects from a transcript by reading it.

Reading transcripts does more than convince the reader about the need for analysis tools. One immediately notices certain patterns of command usage, i.e., a sequence of commands that the user repeats. A major contribution of this research is a new way to extract information from transcripts: an algorithm which scans transcripts to detect repeating patterns.

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 input actions performed by the user. In a command line interface this extract consists of lines of command strings. Other interface styles, such as direct manipulation [17], will have different types of user actions. Such actions, however, can be represented in some textual fashion [19], and thus should yield to the techniques described here. It is therefore sufficient to examine only command line interfaces as a preliminary investigation.

Repeating Patterns: The Repetition Hypothesis

This paper will assume that the Rationality Principle [3] holds, i.e., that user behavior at a computer is purposeful, and hence that 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, 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. 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

UNIX is a trademark of Bell Laboratories.

are partly governed by the interface, the quality of the user's interaction with the system is also reflected by those transcripts.

In order to detect user tasks, this research hypothesizes that *repeated* sequences of actions in a transcript indicate a task rather than a random sequence of user actions. Moreover, the greater the number of repetitions of a sequence, the greater the likelihood that the sequence actually represents a task. Repeated sequences of user actions are therefore behaviorally interesting.

Repetition is also interesting of itself. Each action a user performs takes a certain amount of time and involves a chance for errors. Repeating such actions increases both the performance time and the chances for errors. By detecting frequently repeating actions and providing macros for them, it may be possible to reduce performance times and errors.

When errors are generated during the course of repeated actions, it may be the case that users are having problems with a particular task. Users may be trying variations of a command (e.g., changing the order of command arguments) in an attempt to make it work. Such repetitions might indicate problems with, for example, the command syntax or help system of the interface.

This paper therefore hypothesizes that repeated sequences of user actions may indicate problems with the user interface. The ability to detect such repeating patterns may therefore prove useful to interface evaluators.

The String Model of Transcripts

The problem of detecting repeating sequences of command strings in the extract is equivalent to detecting repeated 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 repeated 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 from a purely syntactic point of view, 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 other substrings as required, and not be inundated with data. The detection algorithm is therefore confined to *maximal repeating patterns*, which are now defined. For a formal definition, see [18].

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. Substrings of longer patterns are also considered MRPs if they occur independently. 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, i.e., 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 the "x"). This special case is an attempt to preserve "context," i.e., the sequence "abc" has two contexts in which it occurs, "abcd" and "abce." 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).

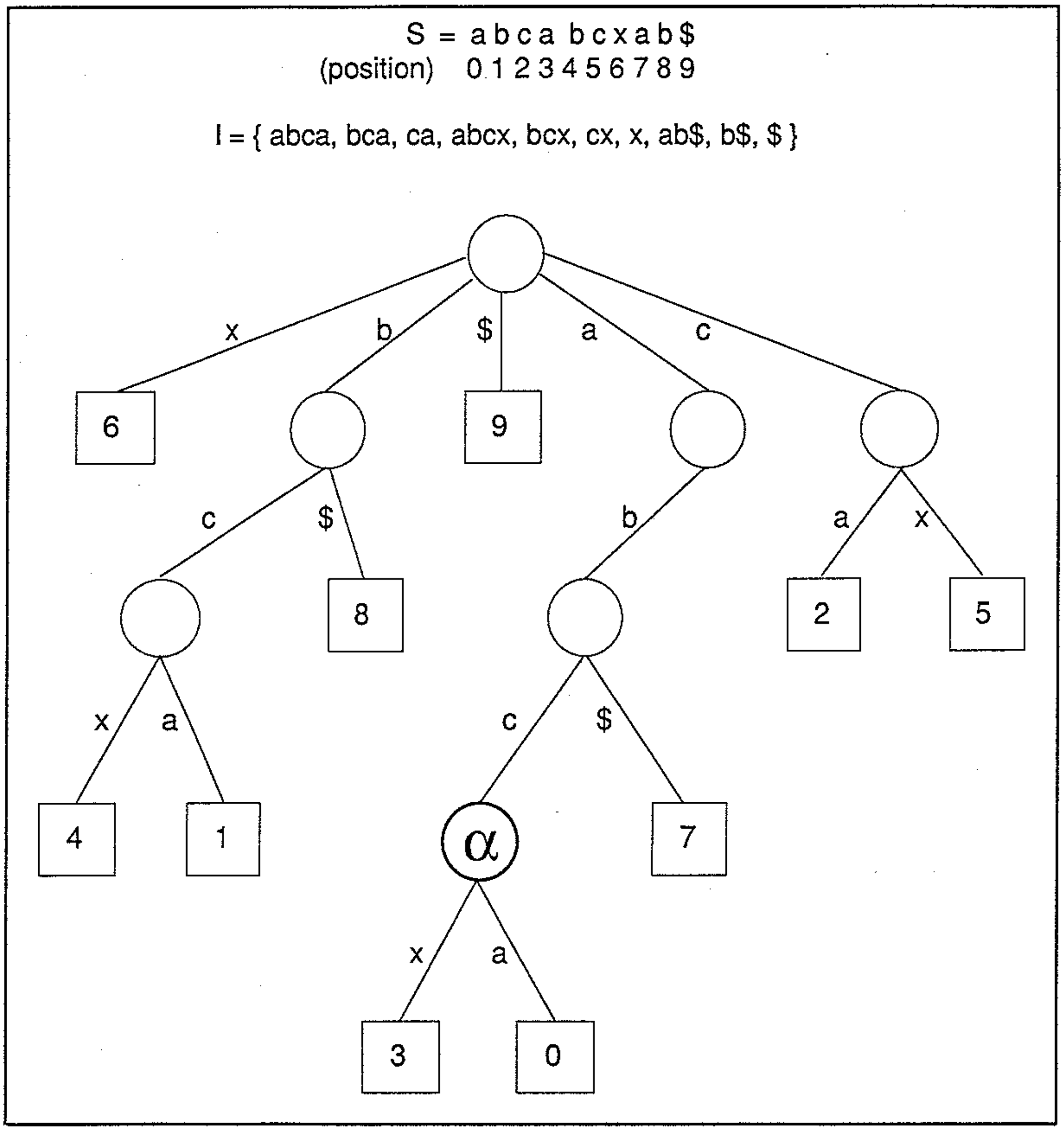
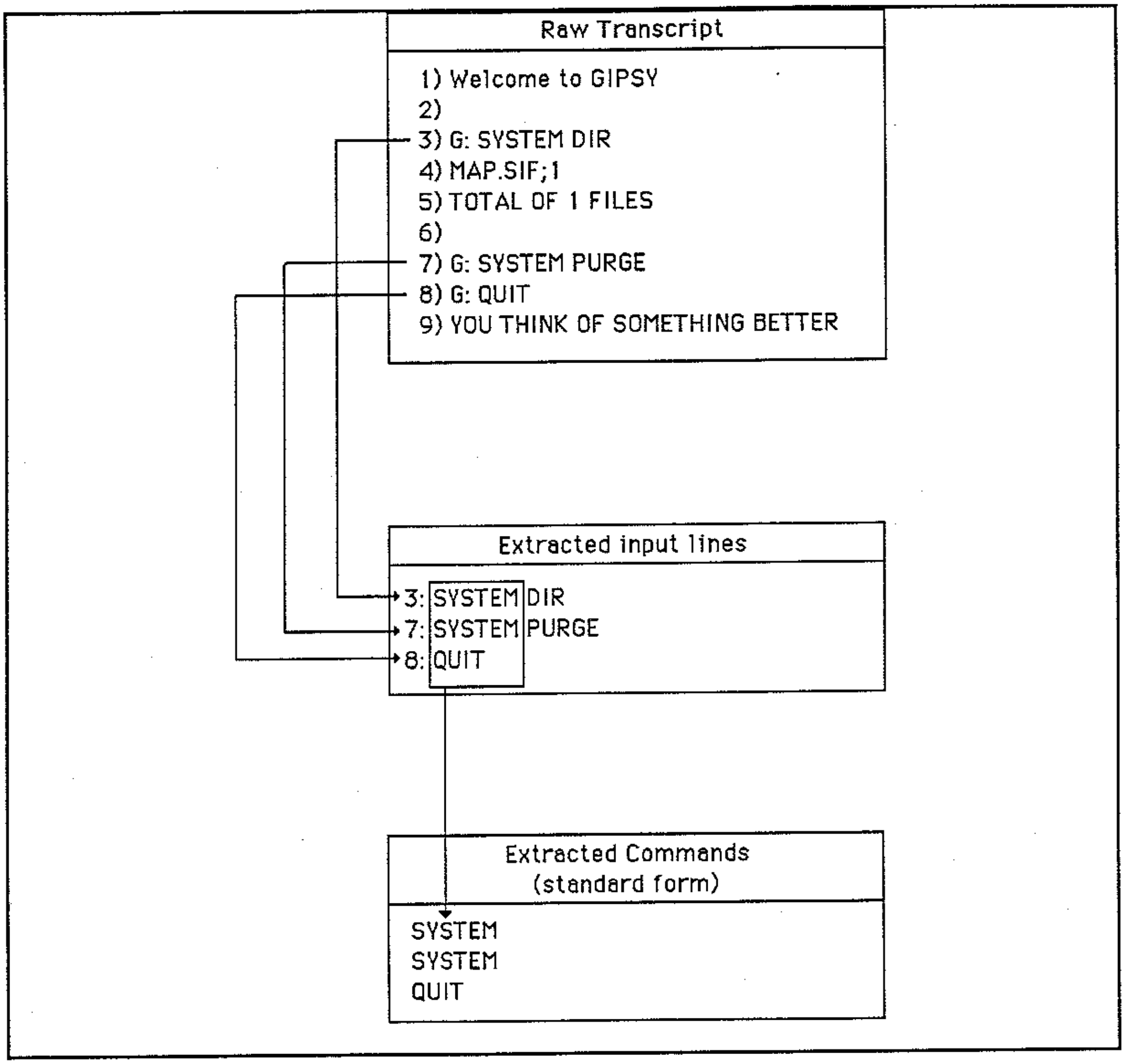

S = a b c a b c x a b $
(position) 0 1 2 3 4 5 6 7 8 9
I = { abca, bca, ca, abcx, bcx, cx, x, ab$, b$, $ }
x
b
$
a
c
6
9
c
$
b
a
x
8
2
5
x
a
c
$
4
1
α
7
x
a
3
0

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

Algorithm

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

Siochi [18] has developed an order O(n^2) algorithm to detect all the MRPs in a transcript, as well as report the positions at which they occur. This 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



```
1) Welcome to GIPSY
2)
3) G: SYSTEM DIR
4) MAP.SIF;1
5) TOTAL OF 1 FILES
6)
7) G: SYSTEM PURGE
8) G: QUIT
9) YOU THINK OF SOMETHING BETTER
```

```
3: SYSTEM DIR
7: SYSTEM PURGE
8: QUIT
```

```
SYSTEM
SYSTEM
QUIT
```

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

that position and nowhere else. The path from the root to a leaf corresponds to this identifying substring. As a result, any proper substring 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 proper 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 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.

TOOLS

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, the 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 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 [12, 13], a pair of compiler writing tools available on UNIX systems. LEX and YACC effectively transform a grammar describing the raw transcript file into a normalizer.

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)
- 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 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 – the complete transcript data file containing both the user input and the system output
- .inp – derived from the .raw file and contains only those lines from the .raw file that correspond to user input
- .cmd – derived from the .inp file and contains only the command portion 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 of 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). Each MRP in the list has at least two instances that occur at distinct positions in the transcript file[1]. An interface evaluator analyzes the transcript by examining the 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 first 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 interesting 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.

[1] If a sequence of commands occurs at only one position, it does not repeat (hence is not 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.

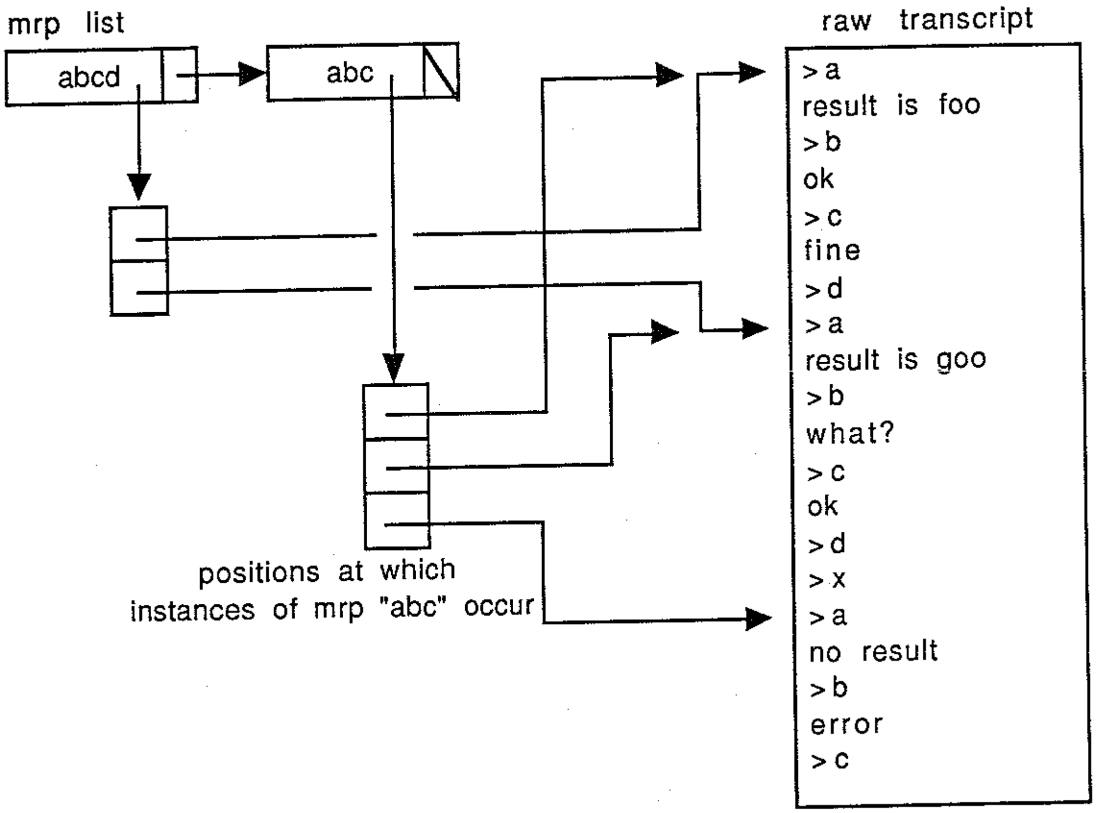


Figure 3. Relationship of MRP list to the raw transcript file.

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.

TESTING MRP USEFULNESS

The usefulness of MRPs in interface evaluation was determined 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 its 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 into the method should be undertaken.

GIPSY

GIPSY is an image processing system designed to run on the VAX™ series of computers [7]. 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.

Data Collection

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

VAX is a trademark of Digital Equipment Corporation

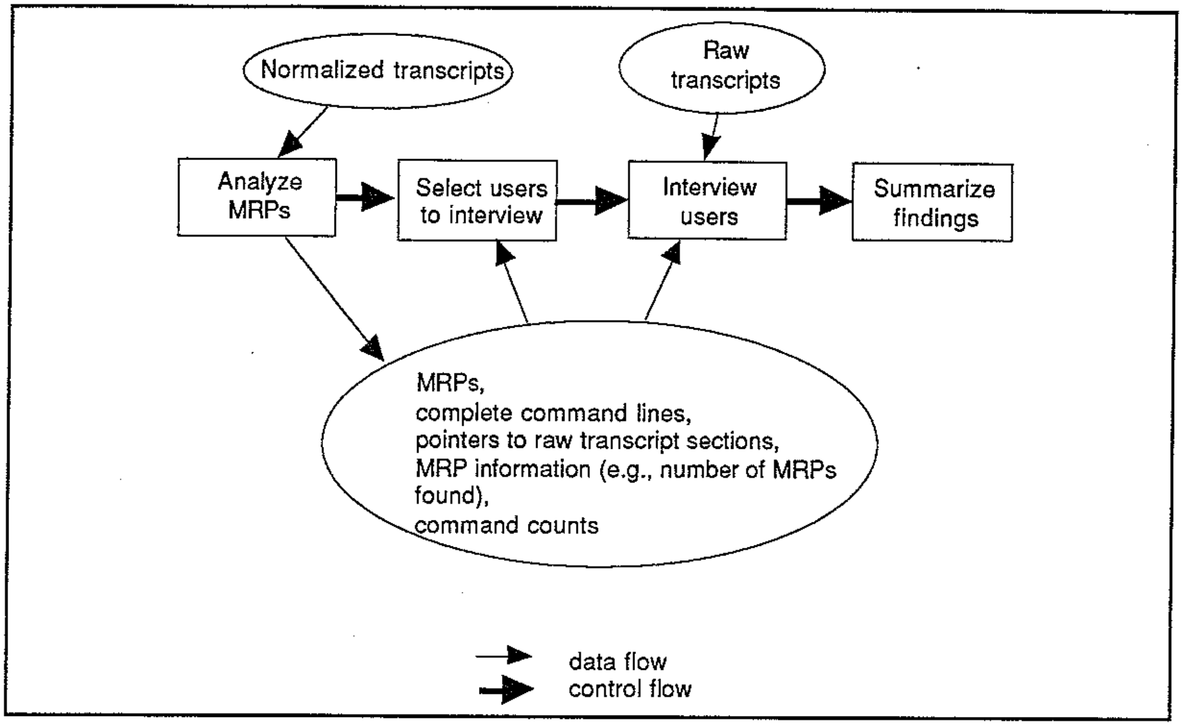


Figure 4. The Evaluation Procedure.

months, producing over twelve and a half megabytes or three hundred thousand lines of raw transcript.

Data Analysis

The evaluation procedure is illustrated in Figure 4, while Figure 5 shows the use of the tools in the procedure. In this analysis phase, the first step was to concatenate in chronological order the collection period transcripts of each user into a single raw transcript representing all the sessions of that single 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 using the MRP tool. The MRP list was scanned by the evaluator, and where necessary, the complete command lines corresponding to the MRPs were reviewed and the raw transcript sections corresponding to the MRPs examined.

At this point a fair number of GIPSY problems were detected. However, several MRPs appeared anomalous or insignificant. These were investigated by interviewing two GIPSY users. The evaluator generated a list of questions to ask selected users. The users were asked to validate the deductions made by the evaluator, or to explain certain episodes in the transcript which were indicated as interesting by the MRP tool but which appeared anomalous to the evaluator. This step was essentially a debriefing structured by the results of the MRP analysis. The findings were then summarized.

RESULTS

The MRP analysis showed several interesting GIPSY interface problems. The interviews validated certain deductions made in that analysis and clarified the anomalies of that stage. In all, twelve problems were revealed, two of which were previously known.

MRP Tool Results

The transcripts of seventeen users, totaling 17,086 command lines, were analyzed over an eight hour period. The bulk of this time was spent studying the lists of MRPs generated by the tool, each of which took a few seconds on average to generate. Table 1 gives, for each user, the number of MRPs detected, the length of the longest MRP, and whether or not both type 1 and type 2 MRPs were found.

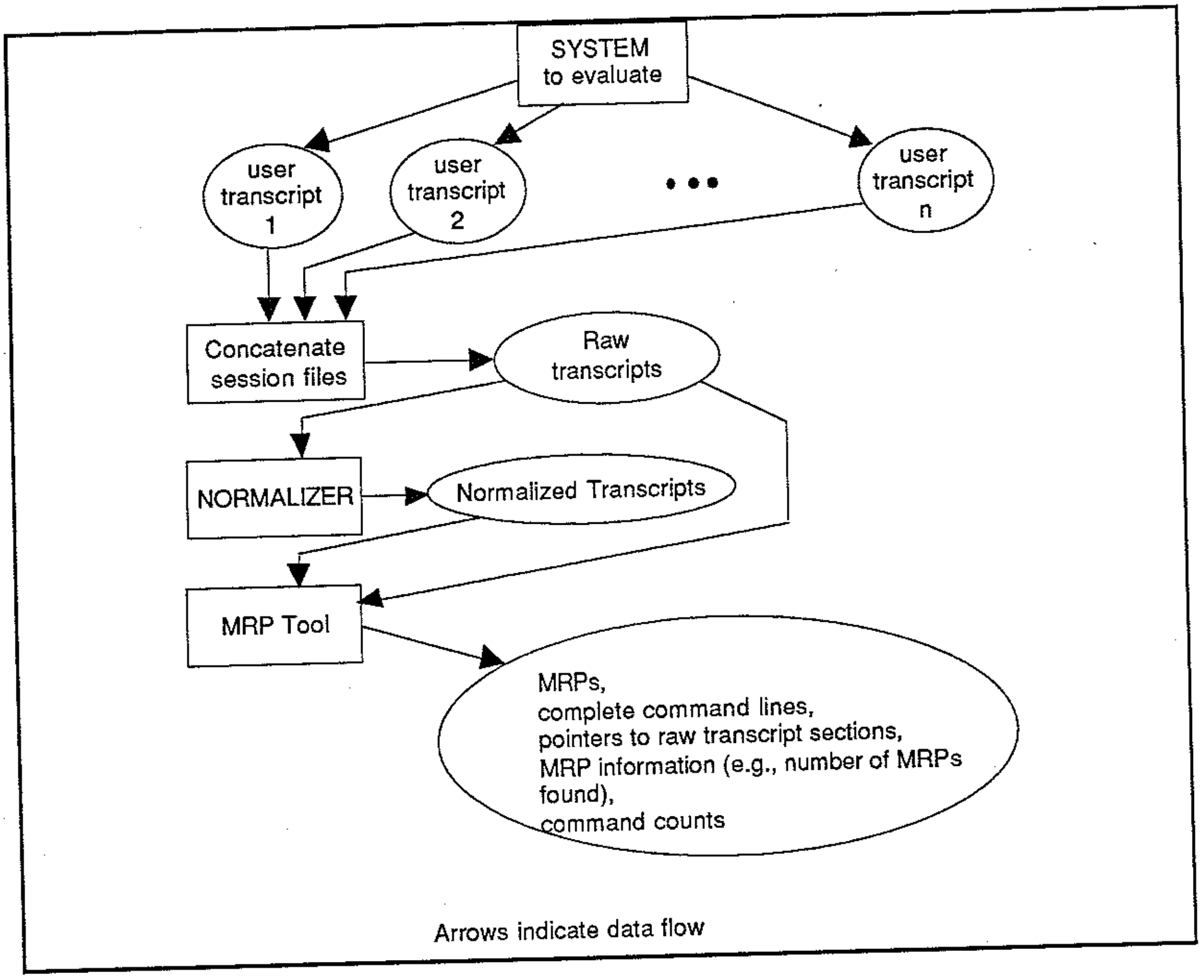


Figure 5. MRP analysis tools and the evaluation process.

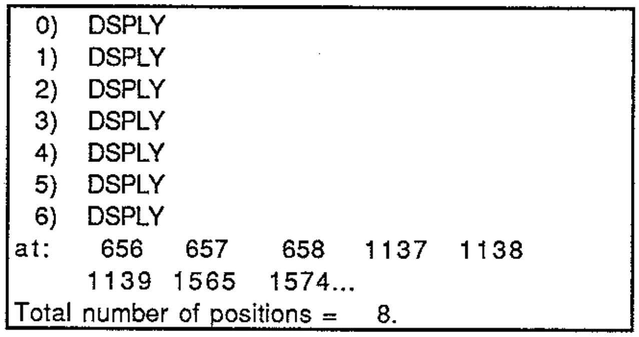
Table 1. Some information from MRP analysis of seventeen transcripts.

User	Number of Lines	Number of MRPs	Length of longest MRP	Type 1 MRP ?	Type 2 MRP ?	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 1 MRP
47% of users had type 2 MRP

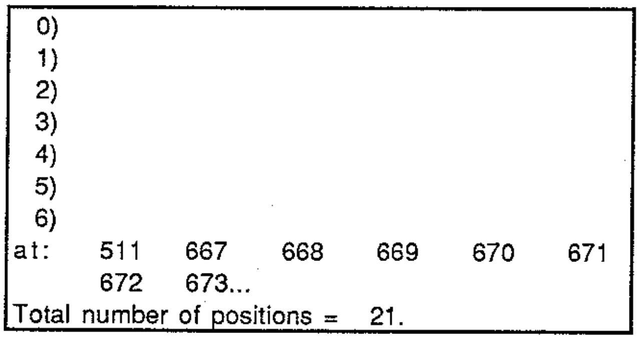
Some of the MRPs were considered noteworthy because they violated expected usage patterns of commands in general. For example, ten GIPSY commands were found in MRPs consisting of consecutive invocations of the same command (see Figure 6). These MRPs are the type 1 MRPs reported in Table 1. Type 1 MRPs may indicate that a user needs to perform the same command on several objects, 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, or a user may be debugging a GIPSY macro. In the first case, a possible remedy could be to allow an arbitrary number of arguments for each such command. The second case demands a closer study of the nature of the "fine-tuning." The fact that 82% of the sample users exhibit this MRP type suggests that this indicates a problem inherent in the interface design, rather than a collection of user idiosyncrasies.



```
 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.
```

Figure 6. An instance of a type 1 MRP: consecutive invocations of the same command.

Type 2 MRPs consist of consecutive lines where no commands were entered (see Figure 7). These indicate anomalous use of 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. It may also be due to objects falling on the keyboard and striking the return key. However, the high proportion of users who exhibit this MRP, and the long experience designers have with keyboards suggest that system response time is the more likely cause.



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

Figure 7. An instance of a type 2 MRP: empty command lines.

Other MRPs provided specific information about which commands were used together. An example is described below. The notation "X A > B" means that the user invoked command X using file A as input and file B as output.

```
Occurs 235 times:
CHRSIF A > B
EXSIF  B
DSPLY  B
```

This MRP indicates that a user tends to modify a file (using EXSIF) that was just processed with the CHRSIF command. The DSPLY (display) command is used to verify the modifications. Because this MRP occurs many times (235 times in three months), a macro which combines these commands is desirable.

Such MRPs often use the output of one command as input to a succeeding command. These MRPs suggest specific macros, including any necessary parameters and local variables.

Several MRPs show the user confirming the effects of an image manipulation command. They indicate the strong need for feedback in the interface. Instead of typing the DSPLY command each time, the image manipulation commands could have an option to redisplay the image after processing it. This would cut down on the amount of user typing required.

```
Occurs 29 times:
STOP
STOP
```

This MRP is highly unusual because it shows that in several sessions users did not invoke any GIPSY commands, i.e., users would start GIPSY then immediately quit. It is likely that users forgot to perform some command in the local operating system, or decided to do something else. However, GIPSY provides a command (SYSTEM) which allows users to access the local operating system from within GIPSY. Given the existence of this command, the "stop; stop" MRP becomes all the more anomalous.

Structured Interview Results

After the studying the MRP tool results, it became clear that some MRPs could only be explained 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., the students in the image processing class.

In each interview, the user was presented with 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 the MRPs alone, neither user could remember what they 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 the raw transcript corresponding to the MRPs, both users remembered almost all the tasks. This is a remarkable finding, considering the time that had elapsed.

The interviews revealed or confirmed several specific problems both in the GIPSY interface and the functionality of some commands. For example, the deduced reasons for the "stop; stop" MRP above were confirmed by both users. In fact, 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, then return to GIPSY, than to use the GIPSY SYSTEM command.

In another MRP, it was deduced that U14 was using a command to test for the length of a file. U14 performed a binary search by successively specifying values to the command depending upon whether an error (attempt to access a non-existent record) occurred. The command that provides information about files did not report its length. This led to the binary search behavior of U14. When U14 was questioned about this, he verified the deduction of a binary search and its use to determine a file's length.

Another interesting MRP from U14 was

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

(ARITHM and EXSIF modify files, while LWPIC prints them out.)

When questioned about this MRP, U14 reported that the LWPIC command did not support the printing of binary image files. Note that although this MRP did identify the need for a macro, in this case the macro was *compensatory* behavior. This result makes the point that MRPs identify repeating usage patterns, but not the *reasons* for the repetition, nor solutions to the problems identified by the repetition. Examination of the contexts in which the MRP occurs is required to deduce those reasons.

MRPs also 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 so clear as those returned by the operating system: "It takes you a while to realize what's wrong."

Problems With the Technique

A problem with the technique was that the number of detected MRPs reach into the hundreds for almost half of the transcript files (see Table 1). If this were the case in general, the problem of tedium is resurrected. Figure 8, 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 initially so much information.

A related problem with the detected MRPs 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 the MRPs, i.e., reporting only those

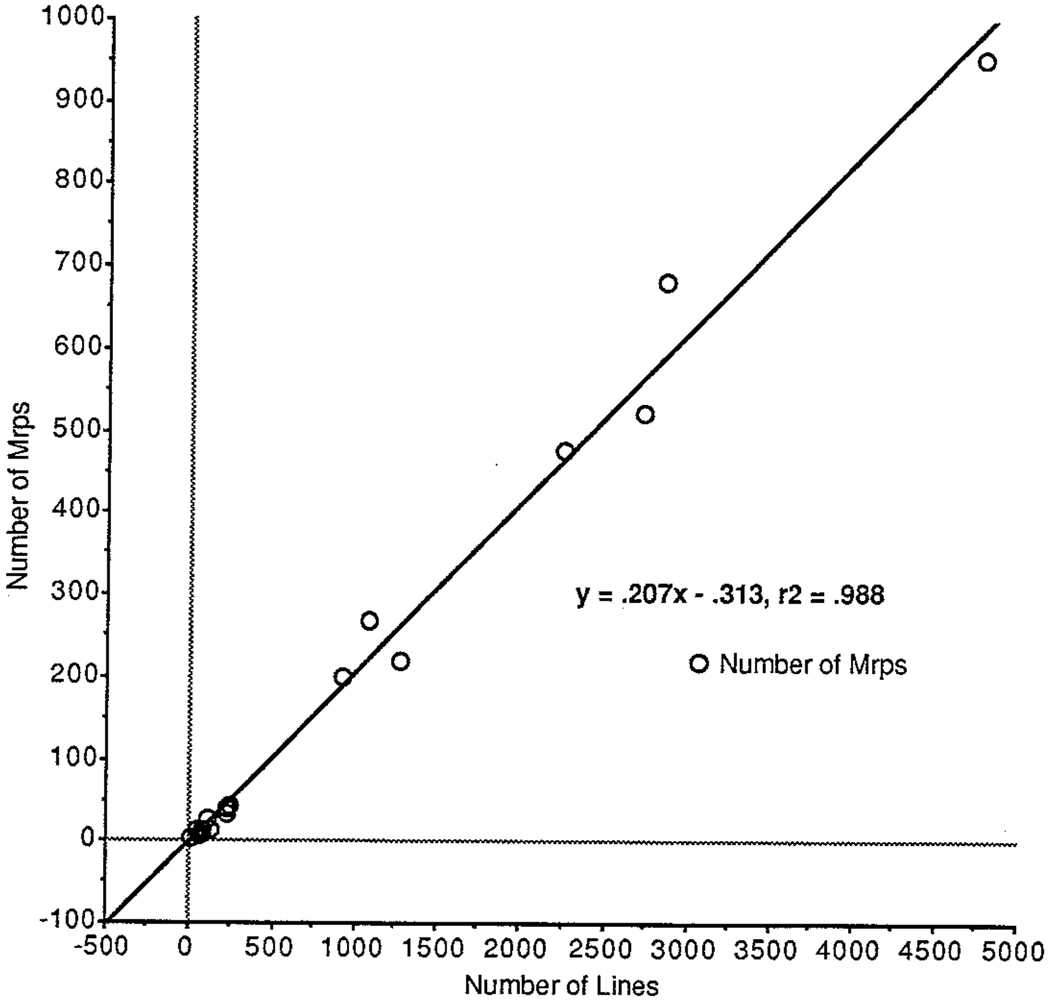


Figure 8. Plot of number of lines vs. number of MRPs detected.

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 9).

CONCLUSIONS

The results demonstrate that the technique was useful for finding specific problems in the GIPSY interface, e.g., the problems with the SYSTEM command. The MRP algorithm found repeating patterns of user actions in the transcripts, and these patterns indicated aspects of the GIPSY user

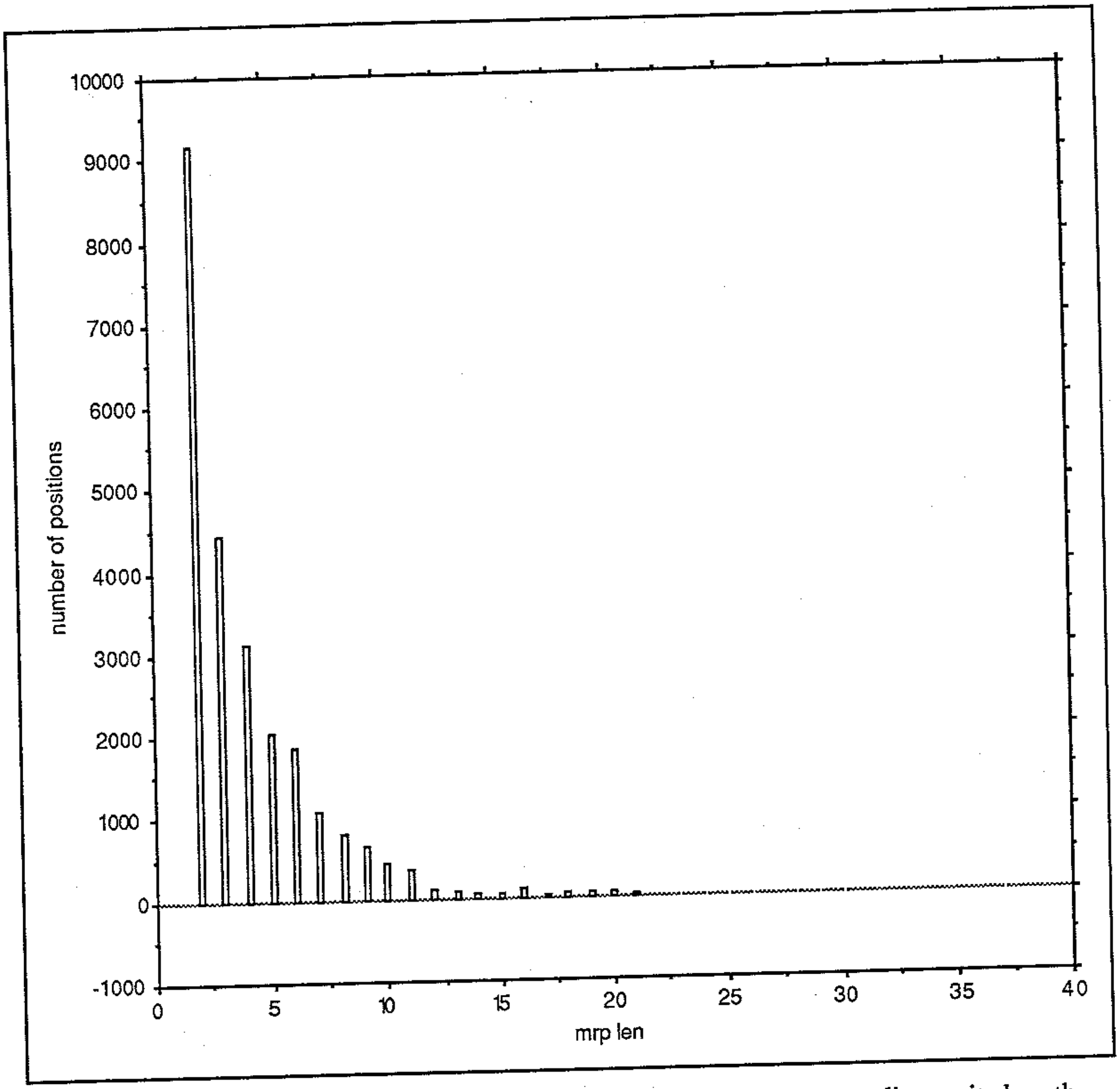


Figure 9. Distribution of number of positions an MRP occurs at according to its length.

interface which needed attention. Although the MRPs did not show the root cause of a problem, much less indicate solutions, the MRPs did identify specific, real problems. It is therefore reasonable to expect that this technique would work for other command line-based systems.

Advantages

The most important benefit provided by this technique is its rapid ability to identify sections of the raw transcript which potentially show difficulties users were experiencing with the system. This means that vast amounts of data can be analyzed in a short time. The three calendar months of *actual* use data of seventeen users, totaling seventeen thousand command lines were analyzed in about eight hours. Of this total, the MRP tool took only a few seconds per transcript to detect the MRPs.

The technique was also useful in preparing and conducting the structured interviews. Complete command lines and the sections of raw transcript identified by the MRPs enabled users to recall what they had been doing, and reminded them of problems they had with the system. Although for the interview the resulting MRPs had to be studied (taking about one and a half hours per MRP list) and users debriefed (taking about one and a half hours each), this still represents a tremendous savings of time if the equivalent observations had been made by videotape.

Another asset of this technique is the quick access the MRP tool provides to information such as MRPs, complete command lines, and raw data. The interviews depended extensively on this information, and when a busy user has graciously volunteered some time, it is important that the user not be kept waiting.

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.

Limitations

The central limitation of the technique is the type of information it provides. The MRPs focus on specific, detailed problems encountered by users. General aspects of the usability of the system are not directly exposed by this technique. 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 finding out 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 the MRPs alone does not produce as much insight as studying the complete command lines. This is because MRPs show only command names, while complete command lines show arguments as well. Similarly, the interviews generated more information about the interface. 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 of the detected MRPs showed classes of usage patterns such as repeated invocations of a command on an object, or pipelining the output of one command into the input of another. A broader evaluation tool should encompass these diverse elements.

Another characteristic of this technique is the need for analytical acumen. The analysis depends on the evaluator's skill and knowledge of both the evaluation method and the application system being evaluated. Human judgement is still required in deciding which MRPs to examine further, in making deductions, and in proposing changes. Also, a sequence of commands may be repeated often, but 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 characteristic of the interface.

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.

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.

FUTURE WORK

Reducing Analysis Time

The MRP tool provided detailed information about interface problems by pointing to transcript sections that were analyzed by both the evaluator and two users. This is valuable information, but considering the graph of Figure 8, a large scale study would inundate the evaluator with MRPs. This can be addressed by reducing the number of MRPs to study.

One way to accomplish this would be to limit the amount of data collected, as is eventually done. The maximum amount of data that would be collected can be estimated by first determining how many MRPs the evaluator can analyze in the time scheduled. This number is then substituted for the variable y in the equation shown in Figure 8, and that equation solved for x, the number of command lines a user types.

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. Algorithms which reduce the amount of redundant information, and a re-evaluation of the MRP definition (specifically removing the independent occurrence condition) should be investigated.

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.

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 with another means of selecting which MRPs to focus on. MRPs with elapsed times greater than expected would be prime candidates for further investigation. Time information can also be used in a cost-benefits analysis to determine which problems in the interface to fix. This is an extremely important benefit in any engineering process, for as Whiteside, Bennett, & Holtzblatt [21] point out, engineering a piece of software involves the allocation of scarce resources.

Similarity Indicator for Command Lines

MRPs show only commands. It was noted that more information, in the form of the complete command lines (i.e., MRP instances), was necessary for analysis. A means of representing those portions of the 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 the question marks indicate differences in the MRP instances. This representation technique could be implemented using a modified string to string correction algorithm, similar to that used by the *diff* program in UNIX.

Use as a Support Technique

Another interesting avenue to explore would be the use of MRP detection 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 method points to potential problem sections in the transcript, it could also be used to indicate similar sections of videotape. This may be readily accomplished by inserting the frame numbers generated by the videotape machine into the transcript file at the appropriate locations

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.

The advantage of the technique is its speed and ability to scan large amounts of data, providing pointers to transcript sections which potentially show problems users were having. 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 these limitations, the technique provided useful information about the GIPSY 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.

ACKNOWLEDGEMENTS

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 thank the SDA Lab at Virginia Tech for serving as a testbed for the research. Finally, we would like to express our appreciation to the Contel Technology Center for funding our current research on this topic.

REFERENCES

1. Aho, A., Weinberger, P., and Kernighan, B. AWK — a Pattern Scanning and Processing Language. *Soft. Prac. and Experience.* (Jul. 1978).

2. Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Series in Computer Science and Information Processing. Harrison ed. Addison-Wesley, Reading, Massachusetts, 1974.

3. Card, S. K., Moran, T. P., and Newell, A. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum, Assoc., New Jersey, 1983.

4. Carroll, J. M., and Rosson, M. B. "Usability Specification as a Tool in Iterative Development." *Advances in Human Computer Interaction.* Hartson ed. Ablex, Norwood, New Jersey, 1985.

5. 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, California, 1983, pp. 897-900.

6. Ehrich, R. W. *The DMS Multiprocess Execution Environment.* CSIE-82-6, Virginia Tech Computer Science Dept., 1982.

7. Garland, E., and Ehrich, R. W. *A GIPSY Primer.* Spatial Data Analysis Laboratory: Blacksburg, VA., 1987.

8. 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* ACM, New York, 1985, pp. 93-97.

9. Gould, J. D., and Lewis, C. Designing for Usability: Key Principles and What Designers Think. *Commun. ACM. 28* (1985), pp. 300-311.

10. Hanson, S. J., Kraut, R. E., and Farber, J. M. Interface Design and Multivariate Analysis of UNIX Command Use. *ACM Trans. Office Info. Sys. 2*, 1 (1984), pp. 42-57.

11. Hartson, H. R., and Hix, D. Toward Empirically Derived Methodologies and Tools for Human-Computer Interface Development. *IJMMS. 31* (1989), 477-494.

12. Johnson, S. C. *Yacc: Yet Another Compiler Compiler*. Computing Science Technical Report No. 32, Bell Laboratories, Murray Hill, NJ 07974, 1975.

13. Lesk, M. E. *Lex — A Lexical Analyzer Generator*. Computing Science Technical Report No. 39, Bell Laboratories, Murray Hill, NJ, 1975.

14. Mackay, W. E.et al. 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). ACM, New York, 1988, pp. 133-137.

15. 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). North-Holland, Amsterdam, 1983, pp. 78-82.

16. 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). ACM Press, 1988, pp. 102-108.

17. Shneiderman, B. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Comput. 16*, 8 (1983), pp. 57-69.

18. Siochi, A. C. *Computer-based User Interface Evaluation by Analysis of Repeating Usage Patterns in Transcripts of User Sessions.* (Dissertation) Virginia Polytechnic Institute & State University, Blacksburg, Va., 1989.

19. 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). ACM, New York, 1989, pp. 183-188.

20. Weiner, P. Linear Pattern Matching Algorithms. In Proceedings of *IEEE 14th Annual Symposium on Switching and Automata Theory* , 1973, pp. 1-11.

21. 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).

22. Williges, R. C. The Use of Models in Human-Computer Interface Design. *Ergon. 30*, 3 (1987), 491-502.

23. Williges, R. C., Williges, B. H., and Elkerton, J. "Software Interface Design." *Handbook of Human Factors*. Salvendy ed. Wiley, New York, 1987.

APPENDIX
MRP Tool Functions

- **mkmrp** FILENAME: makes the MRP list from the file named as argument

- **info** : show information about the current MRP list

- **stats** : show command usage statistics

- **show** X | first | next | prev | NUMBER | all | “”: types the MRP marked with X, the first, next, or previous MRP, the nth MRP, all MRPs, or the current MRP.

- **filter** (num | len < | = | > NUMBER): filters the MRP list by number of occurrences or length of MRP

- **details** first | next | prev | all | “”: types the command lines corresponding to the positions of the current MRP. arguments are similar to the show command

- **wid** NUMBER: set the number of lines to print before and after each MRP detail

- **raw** : view the raw transcript file

- **mark** ? | X <COMMENTS>: list the marks, or mark the current MRP with the character X and comments

- **go** main | filter: goes to the main MRP list or the filter list

- **save** <all> mrp | details | info | stats FILENAME: saves the current MRP, details, information, or statistics to FILENAME. if “all” is specified, saves all MRPs, etc. *replaces* contents of existing file.

- **append** <all> mrp | details | info | stats FILENAME: same as save, except appends to FILENAME

- **readfrom** <FILENAME>: reads MRP tool commands from FILENAME. if argument is missing, reads from the keyboard

- **!** unix command string: shell escape; argument is a command line which is sent to UNIX for execution

- # <COMMENTS>: use this as first word on a line to mark the rest of the line as a comment.

- **quit** : exit the MRP tool