

A methodology for the capture and analysis of hybrid data: a case study of program debugging

Article (Unspecified)

Romero, Pablo, Cox, Richard, duBoulay, Benedict, Lutz, Rudi and Freudenberg, Sallyann (2007) A methodology for the capture and analysis of hybrid data: a case study of program debugging. Behavior Research Methods, 39 (2). pp. 309-317. ISSN 1554-351X

This version is available from Sussex Research Online: <http://sro.sussex.ac.uk/1744/>

This document is made available in accordance with publisher policies and may differ from the published version or from the version of record. If you wish to cite this item you are advised to consult the publisher's version. Please see the URL above for details on accessing the published version.

Copyright and reuse:

Sussex Research Online is a digital repository of the research output of the University.

Copyright and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners. To the extent reasonable and practicable, the material made available in SRO has been checked for eligibility before being made available.

Copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational, or not-for-profit purposes without prior permission or charge, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

A methodology for the capture and analysis of hybrid data: A case study of program debugging

PABLO ROMERO, RICHARD COX, BENEDICT DU BOULAY,
RUDI LUTZ, AND SALLYANN BRYANT
Sussex University, Brighton, England

This article describes a methodology for the capture and analysis of hybrid data. A case study in the field of reasoning with multiple representations—specifically, in computer programming—is presented to exemplify the use of the methodology. The hybrid data considered comprise computer interaction logs, audio recordings, and data about visual attention focus. The capture of the focus of visual attention data is performed with software. The software employed tracks the user's visual attention by blurring parts of the stimuli presented on the screen and allowing the participant to see only a small region of it at any one time. These hybrid data are analyzed via a methodology that combines qualitative and quantitative approaches. The article describes the software tool employed and the analytic methodology, and also discusses data capture issues and limitations of the approach.

Digital technology enables us to capture a variety of data in empirical studies. Besides video and audio, it is now possible to record, for a single experimental session, both visual attention and a detailed account of other user-computer interaction events.

Throughout this article, we refer to this mixture of data of different types that originate from the same empirical episode as *hybrid data*. These are rich data that can comprise qualitative and quantitative elements. The analysis of such data requires us to coordinate their components in such a way that they integrate to provide a single, coherent story of the episode observed. The analysis of hybrid data has been a method used to overcome the weaknesses or biases of approaches based on single data types (Denzin, 1997). Computerized tasks offer a good opportunity to capture hybrid data, because the user-computer interaction can be logged, verbalizations can be digitally recorded, and performance data can also be registered.

We are particularly interested in computerized tasks in which the user has to interpret and coordinate multiple representations presented on the computer screen. These include a wide variety of tasks because almost any computerized activity performed using a graphical user interface involves working with multiple windows. Specifically, our focus is on understanding the coordination of representations in reasoning and learning (Ainsworth, 1999; de Jong et al., 1998), particularly during troubleshooting-based problem solving. Troubleshooting within a computerized environment is an activity particularly suitable for studying representation coordination because such environments normally provide multiple representations that are concurrently displayed, adjacent, dynamic, and linked. Computerized environments for

troubleshooting normally simulate (and enable users to monitor) the system under inspection by offering a set of visualizations that change over time and that give a detailed account of the behavior of the system at the level of its internal structure. In this way, the user can execute simulations for specific cases (stopping at predefined moments of this execution if necessary), observe how the state of the system changes through time, and detect the elements responsible for faulty behavior.

Performing troubleshooting activities within a computerized environment is a task particularly suited to capturing process data. Such data can include (among other information) records of (1) the interaction events the user performed in order to control the execution of the simulation, (2) the windows and visualizations employed, and (3) the user's verbalizations. Computerized troubleshooting environments can be complemented with the appropriate functionality to capture these sorts of data.

In this article, we describe a methodology for the capture, analysis, and synthesis of hybrid data focusing on the coordination of multiple representations when working in computerized environments. We present an example demonstrating the application of this methodology in the area of software troubleshooting (program debugging) by novice programmers.

The capture and analysis of digital hybrid data are particularly relevant to the study of program debugging because programmers normally work within a computerized environment to perform this task. The study of debugging is of particular importance in the case of novice programmers, given that they often spend a large amount of their time dealing with errors in their code. To detect these errors, novices normally try to coordinate and make sense of

P. Romero, pablror@sussex.ac.uk

the program code, the output, and a set of dynamic visualizations that illustrate the state of the program at different moments of its execution. Studying novices' debugging behaviors can offer useful information about the development of programming skill, common programming misconceptions, and efficient debugging strategies.

The article comprises six major sections. The first of these presents a description and analysis of the troubleshooting activity and the role of external representations within it. Next, we present a brief review of related methodologies for the capture and analysis of hybrid data for reasoning and learning, and specifically for the case of computer programming. In the subsequent three sections, we describe our methodology for data capture and analysis. Finally, the Data Capture and Further Analyses section presents a discussion and critique of the approach and suggests possible lines of further research.

Representation Coordination in Troubleshooting Tasks

In this article, we describe an approach to hybrid-data capture and analysis that we believe is applicable to studying a wide range of fault-finding activities. When troubleshooting is performed with the help of a software tool, the user is frequently presented with a simulation of the faulty system. This simulation normally presents several aspects of the system under inspection using a set of dynamic representations. An important task for the user is therefore the interpretation and coordination of a multirepresentational system. Examples of this situation are found in applications as diverse as finding faults in industrial machine tools (Kranzlmüller, Grabner, & Volkert, 1997), in parallel and distributed computer software (Gabbay & Mendelson, 1999; Marzi & John, 2002), in large marine engines (Hountalas & Kouremenos, 1999), in integrated-circuit hardware designs (Friedrich, Stumptner, & Wotawa, 1999), and in aircraft fuel systems (Papadopoulos, 2003), among others.

The fault-finding process can be illustrated in more detail with a concrete example from the area of commercial call-center systems. In this example, when errors are reported, a number of log files must be inspected and coordinated in order to ascertain the nature of the problem and the subsystem in which it occurred. In commercial call-center systems, log files in different formats are sometimes produced by three different subsystems: the telephony subsystem, the user interface, and the software connecting these two.

The telephony subsystem produces a log file with time-stamped entries indicating changes in state of the telephones—for example, when a telephone was in use, was free to take a call, or was blocking calls. The user interface subsystem produces another log file that indicates which keypresses took place in the telephony window that was used to control the telephone system. These two representations allow the programmer to ascertain whether the computer system was “in sync” with the telephone. If this were not the case, some remedial action would be required. The third representation is the trace from the software code. This shows in detail which sub-

outines were called and allows the programmer to build a process-driven mental model of the program's execution. Coordinating and interpreting these representations enables programmers to build a multifaceted understanding of the problem, which in turn supports the fault-finding task.

The interpretation and coordination of dynamic, multi-representational systems are central tasks in troubleshooting; however, empirical studies focusing on these fault-finding activities are rare. The following section presents an overview of troubleshooting studies.

Capturing and Analyzing Hybrid Data

This section presents a brief review of the literature on the most popular types of process data and the methodologies employed for their capture and analysis, particularly in the area of software troubleshooting.

Studies of software comprehension and troubleshooting have tended to capture and analyze performance rather than process data (Davies, 1994; Gilmore, 1991; Gilmore & Green, 1988; Patel, du Boulay, & Taylor, 1997; Vessey, 1989). Such studies have usually captured information about percentages of correct responses and solution times for given debugging tasks. However, when process data have been considered, the most popular types have been verbalizations (Mulholland, 1997; Pennington, 1987; Pennington, Lee, & Rehder, 1995; Vessey, 1985). Experimental participants have been asked to “think aloud,” and their verbal protocols have been analyzed in order to explore strategy and investigate how it relates to programming experience and proficiency (Vessey, 1985); to explore the relationships between notational properties of the language, the computerized debugging environment, and the information types programmers consider important (Bergantz & Hassell, 1991; Mulholland, 1997); and to study program comprehension strategy in terms of the mappings programmers establish between the program and problem domains (Pennington, 1987).

Another type of process data is related to the focus of visual attention, although studies considering this variable have been less frequent. There has been interest in the patterns of visual inspection employed when performing program comprehension tasks. This topic has been investigated either by restricting the environment in such a way that the user's focus of visual attention can be tracked (Robertson, Davis, Okabe, & Fitz-Randolf, 1990) or by employing eyetrackers (Crosby & Stelovsky, 1989). Such studies have analyzed code-reading patterns to investigate whether they are more similar to prose reading or to tasks related to problem solving (Robertson et al., 1990), and also to investigate the relationship between focusing on critical areas of the code and the participant's characteristics, programming experience, and cognitive style (Crosby & Stelovsky, 1989).

An approach that has been even less frequently used is to record user-computer interaction data. Cox (1997), for example, employed this approach to capture and analyze data about representation coordination when constructing and interpreting external representations in analytical reasoning tasks.

Studies that have combined and integrated process data of different types have been scarce in software comprehension and debugging, but the approach has proved very useful for such purposes as evaluating virtual museum applications (Cox, O'Donnell, & Oberlander, 1999), investigating individual differences in the development of logical proofs (Stenning, Cox, & Oberlander, 1995), and studying skill acquisition among health science students learning diagnostic reasoning (Cox & Lum, 2004). Because of the characteristics mentioned above, we believe that the capture and analysis of hybrid data is a suitable methodology for a deep study of troubleshooting activities performed in a computerized environment. The following section describes the computerized environment that, in combination with the appropriate functionality, has been employed for hybrid-data capture in troubleshooting tasks.

The Restricted Focus Viewer (RFV) Technology

The RFV is a program that takes visual stimuli, blurs them, and displays them on a computer screen, allowing the participant to see only a small region of the stimulus in focus at any time (Blackwell, Jansen, & Marriott, 2000; Jansen, Blackwell, & Marriott, 2003). The region in focus can be moved using the computer mouse. In this way, the program restricts how much of any stimulus can be seen clearly. It also records where the unblurred region of the screen is, and so (it is presumed) what the participant is focusing on at any point in time; in this way, the software enables the capture of the moment-by-moment focus of visual attention. Since several RFV stimuli can be present on the computer screen at the same time, the technique enables a user's representation switching between concurrently displayed adjacent representations to be captured for later analysis.

The user interaction data captured via the RFV can be read with Replayer, a companion program that can replay the way an RFV participant moved the focus window over the stimuli. Among its other functions, this companion program can be used as an analysis tool to replay experimental sessions, allowing the researcher to pause, restart, or play back the locus of movement of the unblurred region in real time, or faster or slower.

The RFV records changes in the location of the unblurred region with millisecond precision. The data captured using this program indicate, for each interaction event, the stimulus involved, the elapsed time, the type of event (mouse move or mouse click, for example), and its coordinates on the screen.

An obvious concern is whether blurring most of the screen makes a significant difference in the nature of the task or of the solution process. The RFV has been validated in the context of reasoning about simple mechanical systems via the inspection of static diagrams (Blackwell et al., 2000). That validation study found no significant differences in the inspection strategies of participants who worked with this technology or with eyetracking equipment. One difference, however, was that participants working with the restricted focus technology tended to take more time to perform the task. We return to the issue of validation in the Data Capture and Further Analyses section.

Data Capture: From Single-Type to Hybrid

Recording and analyzing data about the focus of visual attention can offer interesting information about the process of coordinating multiple external representations. However, troubleshooting environments, besides presenting visualizations of the system under inspection, allow users to view the execution of simulations for specific cases. The Software Debugging Environment (SDE) is a modified version of the RFV that incorporates functionality to enable users to view snapshots of the precomputed execution of particular programs. Figure 1 presents a screen shot of the SDE. In this case, the window on the left shows the code of the program. The line of code with the dark background represents the next command to be executed. The other three (blurred) windows show the state of the data structures of the program (top right), the program's output (bottom right), and a subroutine call stack (bottom center); all of them show the same moment in the program's execution.

The SDE also allows the capture of hybrid data. In addition to recording the focus of visual attention, the SDE captures interaction data about the control of the simulation, as well as participants' verbalizations.¹

Presenting the execution of simulations. We have employed the SDE to investigate representation coordination in the area of software troubleshooting (Romero, Cox, du Boulay, & Lutz, 2002; Romero, du Boulay, Lutz, & Cox, 2003; Romero, Lutz, Cox, & du Boulay, 2002). However, the fact that the display of simulations in the SDE is achieved through preconstructed examples enables experimenters to employ it for different domains. The SDE does not perform an actual simulation of the target domain; instead, the experimenter has to prepare a sequence of visualizations depicting states of the system under study at different moments of the simulation for a specific example. The SDE can then present this sequence in steps. The user can move along the sequence by pressing the arrow keys, or see the visualizations associated with the beginning of the execution by pressing the home key, or see those associated with the end by using the end key.

This independence from the target domain increases the generality of the approach. Employing the SDE for other troubleshooting domains would require the modification of the layout and the number of windows displayed; however, the main process, preparing a sequence of visualizations depicting states of the system at different moments of the simulation, would remain unchanged. The layout and number of windows displayed are parameters set in an initialization file, and therefore their modification does not require any change to the SDE. This software system can also be employed in other frequently researched domains, such as e-learning or human-computer interaction. The main limitation of the approach is that, because moving the unblurred spot is done with the mouse, any manipulation of the displayed stimuli has to be performed with a different input device. This would make it difficult, for example, to employ this tool to explore Web site navigation, given that the main form of interaction in this context is clicking on hyperlinks situated inside the displayed windows.

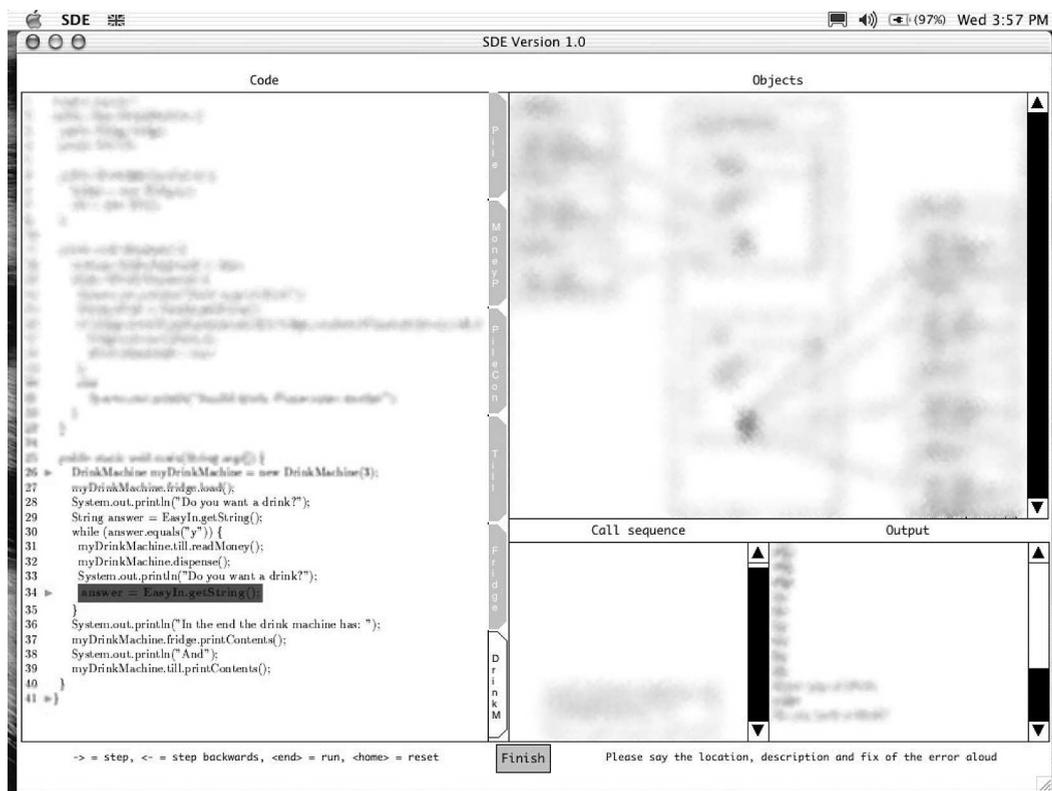


Figure 1. The SDE employed for rich data capturing. The unblurred region is located toward the bottom of the leftmost window.

Presenting visualizations of the state of a system as pre-computed image stimuli creates additional requirements for the SDE. First, these visualizations are sometimes larger than the window in which they appear, and therefore have to be presented within a tab or scrollable pane. A tab pane is particularly useful when presenting text that is organized in a number of files (for example, software modules distributed among several files). A scrollable pane can be used when the images presented are wider or longer than their associated window. Second, presenting a set of concurrent, linked, and adjacent representations encourages participants to switch their focus of visual attention between these representations. This feature thus elicits two different forms of mouse use: The first is the original form, moving the mouse to change the unblurred spot while inspecting a specific window. The second is moving the mouse to switch between windows. To differentiate between these forms of mouse use in the SDE, participants switch between windows by moving the mouse and focus on a specific stimulus region (within a window) by clicking it. On returning to a previously visited window, the region in focus is the one that was set by the previous mouse click performed in that window image.

This last feature makes switching between windows easier, because participants do not have to reestablish the place they were looking at every time they switch attention from one window image to an earlier one (Romero, Cox, et al., 2002). Also, this change enables us to record

(and analyze) window switching and stimulus inspection as two different behaviors.

Capturing data of different types. Besides capturing data about the focus of visual attention (via mouse movements and clicks), the SDE also records information about both the control of the view of the simulated execution and the participants' verbalizations. The keyboard actions that participants perform to control the presentation of the simulation execution (pressing the arrow, home, and end keys) are recorded in the log file generated by this application. In this way, the Replayer program is able to replay, besides a visual focus trace, the execution of the simulation for any particular experimental session. Additional data captured in this log file are related to the extra functionality associated with scrolling and with controlling the presentation of the simulated execution. For the case of scrolling, the additional data are the coordinates (of the stimulus image) located at the top left corner of the scrollable window. For the case of the control of the simulation, the additional data are the ASCII code related to the keyboard actions enabled.

If participants are required to "think aloud," the audio data can be recorded by the SDE. The participants' verbalizations are digitally recorded onto the computer's hard disk in UNIX audio file format (.au, compatible with most PC and Macintosh audio applications) with an 8-KHz frequency rate. In this way, the participants' working sessions with this environment can be replayed "in the

round” for later analysis, and the hybrid data recorded can be analyzed in a synchronous way. This method of hybrid-data capture has been employed in the studies reported in Romero, Cox, et al. (2002); Romero, Lutz, et al. (2002); and Romero et al. (2003).²

Hybrid-Data Analysis

Capturing hybrid data is a technical achievement, but this effort can only pay off if there is a sensible framework for the analysis of such data. The framework we employ allows us to analyze these data both quantitatively and with a methodology that combines quantitative and qualitative approaches. The quantitative analysis relates participants’ performance to their use of the SDE, and the combined approach explores participants’ behavior and strategies by taking into account three types of data synchronically: focus-of-attention trace, control of the presentation of the simulation’s execution, and verbalization data.

Quantitative analysis. The quantitative analysis requires participants’ performance to be extracted from the verbalization data and the log file (of keyboard and mouse actions) to be transformed into a description of simulation execution, window fixations, and switches. Extracting this information from the log file data can be automated (using a computer program that performs this transformation), but the analysis of the verbalization data needs to be performed by human raters.

The description of the simulation execution is an account of how the participants navigate through the different simulation steps. They could view this execution in steps, by moving between predefined points (*breakpoints*) in the simulation. Since this execution is sequential, participants can move forward to the next breakpoint, backward to the previous one, forward to the end of the execution, or backward to the beginning. One question regarding how this execution is viewed is whether there is a relationship between the usage pattern and troubleshooting performance.

Window fixation refers to the total time participants spent focusing on each window of the SDE, and *window switches* refers to the number of changes of focus between the available representations. Relating these two measurements to troubleshooting performance enables us to investigate whether patterns of representation use are associated with different degrees of accuracy.

The content and format of the visualizations presented by the environment can be manipulated. For example, the version of the SDE employed for the studies of representation coordination in debugging reported in Romero, Cox, et al. (2002); Romero, Lutz, et al. (2002); and Romero et al. (2003) presents either graphical or textual visualizations that highlight either data structure or control-flow information. These experimental conditions, together with the variables mentioned earlier (patterns of program execution, representation use, and debugging accuracy) can be analyzed looking for significant main and interaction effects.

Finding out about these relationships offers important information about patterns of environment use and programming expertise. This information can be comple-

mented by investigating the debugging strategies that shaped environment use.

Combining quantitative and qualitative methods. This section describes a methodology to perform an analysis of hybrid data combining quantitative and qualitative approaches. This methodology was inspired by the triangulation approach presented in Denzin (1997) and by the qualitative analysis of verbal data in Chi (1997). To explain this methodology, this section will refer to an example in the area of software troubleshooting.

Our analysis methodology can be employed to explore and explain at a detailed level a specific set of hypotheses. This set of hypotheses could be derived, for example, from the results of a quantitative analysis (e.g., the one described in the previous section). The main idea behind our proposed analysis methodology is to build an interpretation of the participant’s behavior by taking into account all of the available types of data. This interpretation classifies the participant’s behavior into predefined categories associated with the set of hypotheses to be explored. The occurrence frequency of these behaviors can be tested quantitatively in order to provide evidence for (or against) the proposed hypotheses.

The qualitative analysis described here takes into account three types of synchronous data: the trace of the focus of visual attention, the patterns of movement between breakpoints, and the participants’ verbalizations. Because the SDE supports the replay of programmers’ debugging sessions, a rater can execute these replays to extract the desired information. Table 1 presents part of a sample coding sheet. In this section of coding sheet, there are six columns: The first one contains the event number, the second the programmer’s utterances, and the third, fourth, fifth, and sixth the unblurred information displayed by the different windows of the SDE (the Code, Objects, Call sequence, and Output windows shown in Figure 1). Only the contents of the currently unblurred window are shown in the chart. Each row of this table presents one debugging event. In general, debugging events are bounded by pauses or changes of topic in a programmer’s verbalizations (utterances), by interwindow switches of visual attention focus, or by breakpoint switches. The unit of verbalization that we considered an utterance was a verbalization limited by a considerable pause (e.g., one greater than 2 min) and/or by a change of topic. Interwindow switches occurred when the user moved the unblurred area from one window to another, and breakpoint switches took place when the programmer moved the state of the program execution from one breakpoint to another. In the example in Table 1, Event 8 is bounded by a pause or change of topic in the programmer’s utterances, Event 11 is terminated by a window switch, and the rest of the events are bounded by a combination of these two (pauses or changes of topic and window switches).

A basic hypothesis that can be considered to explain and exemplify the classification of behaviors into different categories relates to the possible relationships between debugging performance and behavior. Such a hypothesis suggests that certain program comprehension and debugging strategies are more effective than others. In order to

Table 1
Section of Coding Sheet for a Specific Debugging Session

Event	Verbalization	Code	Focus of Attention		
			Objects	Call Sequence	Output
8	“Enter type of drink. Fanta”	DrinkMachine (line 41)			
9	“Let’s have a look at it again . . .”	DrinkMachine (line 34)			
10	“Ah! Interesting”		piles[0] to piles[3]		
11	“In the Object window, it’s interesting to see . . .”				Enter type of drink. Coke Now enter the number of Fantas. 4
12			piles[0] to piles[3]		

test this hypothesis, a set of coding categories defining different comprehension and debugging strategies has to be defined, and the participant behaviors recorded in the coding sheets have to be classified according to these coding categories.

Program comprehension and debugging behavior coding categories. In order to code the data, a coding protocol had to be developed and operationalized (Chi, 1997). The development of this coding protocol required decisions regarding the number and types of categories to include and the “grain size” of programming event that the coding categories would represent. These decisions in turn depended on the hypothesis being tested, the task, and the content domain. The operationalization of the coding protocol required verification that there was a clear relationship between behaviors in the hybrid data and the coding categories developed. In this step of the process, the effort focused on clarifying definitions of coding categories and resolving possible ambiguities. The development and operationalization phases can be considered top-down and bottom-up processes that are normally applied in an iterative fashion while refining the coding protocol (Chi, 1997).

A partial list of our behavior coding categories is presented in Table 2. This table is divided into categories for program comprehension and program debugging. According to previous research in program comprehension and debugging (Davies, 1994; D etienne, 1997; Pennington, 1987; Pennington et al., 1995; Vessey, 1985), some of these behaviors lead to more success than others. For example, coding category C8 is an instance of a successful program comprehension coding category because it registers the occurrence of a strategy that tries to link the program and problem domains. Such cross-referencing behavior is associated with good debugging performance (Pennington, 1987). On the other hand, program debugging coding category D13 registers the occurrence of a strategy in which the programmer tries to understand specific details of the program without, for example, having a clear idea of what the effects of the error are in terms of the output. Such early preoccupation with program details is associated with poor debugging performance.

Data encoding. To obtain a detailed characterization of programmers’ program comprehension and debugging strategies, their coding sheets are analyzed to look for occurrences of the behavior coding categories in Table 2.

This is a procedure that has to be performed by a human rater and requires that he or she consider information about focus of attention, program execution, and programmers’ verbalizations synchronously. For example, coding category C7 requires verification of whether simple stepping (a program execution behavior) happened while the focus of visual attention was located in a specific window (the Objects view). Verbalizations referring to executing the program in steps and/or to the contents of the Objects window would strengthen the case for the occurrence of this behavior. Table 3 presents an example of data coding for a segment of a debugging session (the coding sheet in Table 1 depicts the same debugging session segment). In this table, Events 9–12 have been categorized as occurrences of the behavior described by coding category C7. Also notice that some events can be coded as instances of more than one coding category. This is the case for Event 12.

Table 3 also includes information about transitions that bounded debugging events. As mentioned in the Comparing Qualitative and Quantitative Methods section above, Event 8 is bounded by a pause or change of topic in the programmer’s utterances, Event 11 is terminated by a window switch, and the rest of the events are bounded by a combination of these two (pauses or changes of topic and window switches).

These low-level behavior coding categories can be integrated into program comprehension and debugging episodes (Vessey, 1985). These episodes are groups of behaviors that have a specific goal in common and that can be used to identify programmers’ strategies.

A cluster analysis allows us to categorize groups of programmers according to their displayed strategies and to compare this categorization with their performance data. The categorization can also be complemented by the findings of the quantitative analysis. In this way, a model of program comprehension and debugging expertise in terms of behaviors and strategies can be empirically derived.

This method for deriving a program comprehension and debugging model, by taking into account several types of data synchronously, has advantages over methods that consider only one type of information. First, the range of behaviors, and therefore of strategies, taken into account by our model can be wider. For example, behaviors C4, C11, and C19 would be difficult to take into account in a model that considers only verbal data. Also, including

Table 2
Sample Comprehension and Debugging Behavior Coding Categories

Comprehension Coding Categories	
C1	Utterances reflect the stage of program execution
C2	Use of breakpoints
C3	Comments relating the information types
C4	Switching between information types a minimum of twice (A to B, then back to A)
C5	Utterances regarding hypothesis followed by switching from code to other type of representation
C6	Utterances regarding hypothesis followed by switching to code from other type of representation
C7	(Single-) Stepping through the code carefully while watching the Objects view
C8	While looking at code or object view, utterances reflect real-world objects in the problem domain
C9	Looking at the Output or Objects view while talking about the code
C10	Utterances relating to higher level entities (e.g., method, a subroutine, a section of code)
C11	Returning to the same line of code from another type of representation several times to understand all its implications
C12	Syntax verbalization
C13	Explaining the code to themselves
C14	Reading the code out loud from top to bottom
C15	Lack of switching between views (especially the code view)
C16	Relating only to real-world objects and only looking at the output
C17	Erratic jumping around within the code
C18	Erratic jumping across information types
C19	Repeatedly examining stereotypical lines of code
C20	Finding a piece of code to account for the output
C21	Searching for a line of code
C22	Paraphrasing as a re-representation
Debugging Coding Categories	
D1	(Single-) Stepping through the code carefully while watching the Objects view
D2	Considering negative evidence in reasoning
D3	Focusing in on an area of code after an uttered hypothesis
D4	Rerunning the code with fix in place
D5	Temporarily considering regions of the program as free from errors
D6	Attempting an a priori classification of errors and acting accordingly
D7	Higher level code browsing to build up a complete picture before testing hypothesis
D8	Being clear that something is a hypothesis
D9	Comparison of actual with expected outcome; early comments suggesting potential causes
D10	Running the whole program again (including the previously commented-out parts) or browsing previously discounted code
D11	Talking in terms of breakpoints (dynamic view) but not stepping through
D12	Utterances of code cliches
D13	Early delving into the details

several types of data enables raters to code particular behaviors with a higher level of certainty (as in the example above about the coding of category C7).

Considering a wide range of strategies in a program comprehension and debugging model could also increase the usefulness of the model. For example, if the model is going to be applied to the design of learning environments for programming (du Boulay, Romero, Cox, & Lutz, 2003), taking into account strategies relating to the focus of visual attention can enable the environment itself, in principle, to provide advice on these matters. The learning environment could, for example, embody a number of monitoring rules that keep dynamic track of both focus of attention and switching behavior, in order to guide students to pay attention in more sensible places.

Data Capture and Further Analyses

This section discusses a number of issues in the practical application of the methodology that has been described. This discussion starts with problems with data capture and moves on to problems with data analysis.

Issues related to data capture involve our assumptions about how the focus of visual attention relates to the tool's unblurred spot, the level of event granularity associated

with the restricted focus approach, and the possible way in which this technology modifies the representation fixation and switching tasks.

The capture of visual attention data assumes that participants are indeed looking at and paying attention to the region in focus within the SDE (and the RFV). This seems to be a reasonable assumption, given that studies validating the restricted focus technology did not find significant differences in the inspection strategies of participants working with this technology and with eyetracking equipment (Blackwell et al., 2000; Jansen et al., 2003; Romero, Cox, et al., 2002). However, similar validation studies for the SDE (Bednarik & Tukiainen, 2004) employing both

Table 3
Section of Data Coding for a Specific Debugging Session

Event	Behavior Category	Transition
8	C22	pause
9	C7	switch/pause
10	C7	switch/pause
11	C7	switch
12	C7, C4	switch/pause

the restricted focus technology and an eyetracker have found that participants also glance at blurred areas with a certain frequency. This behavior could be caused by saccades or could be part of a strategy to minimize explicit interaction with the troubleshooting environment when extracting information that can be inferred, for example, from the generic shape of the (blurred) stimulus image. More experimentation is needed to know which of these explanations is the case.

A related issue is the fact that the level of granularity of troubleshooting events associated with the restricted focus technology is coarser than the one related to, say, eyetracking technology. According to the SDE validation study by Jansen et al. (2003), participants using eyetracking do more representation switching than those employing the restricted focus technology. This difference was not relevant to our studies (Romero, Cox, et al., 2002; Romero et al., 2003; Romero, Lutz, et al., 2002), since the general pattern of interactions and the relative number of switches between representations did not seem to be significantly modified. However, this difference in the level of granularity of the troubleshooting events might be important in other contexts.

The restricted focus technology might also modify the nature of the representation fixation and switching tasks by allowing participants to "bookmark" inspection areas in the visualizations when performing representation switching. The fact that each window "remembers" its region in focus makes window switching easier for participants. Indeed, in the SDE validation study reported in Romero, Cox, et al. (2002), there was a tendency for participants to perform better when working with the restricted focus environment than when working with an environment without a restricted view. One of the possible causes for this tendency is a modification in the fixation and switching tasks.

Unfortunately, taking the bookmark feature away seemed to increase the difficulty of the task considerably, because repositioning the focus area each time there was a window switch required an explicit search episode. Given the fact that the difference in performance reported in Romero, Cox, et al. (2002) was not significant but only a tendency, it was assumed that the modification of the fixation and switching tasks did not influence representation coordination drastically.

Issues associated with the analysis phase include the question of what constitutes a coding event and the need for a predefined set of hypotheses.

Coding events such as interwindow switches of visual attention focus or breakpoint switches are relatively easy to define and identify. However, this is not the case for participants' verbalizations. The unit of verbalization that we have considered an utterance is a verbalization limited by a considerable pause and/or by a change of topic. The first part of this definition is not easy to operationalize, since it depends on the rater's judgment. One way to address this issue would be to employ more than one rater for at least a subset of the participants to verify interrater reliability.

There is also a need for a predefined set of hypotheses. Defining these hypotheses is necessary for the specification of the coding categories for behaviors. An alternative is to perform this specification in a bottom-up fashion, by inspecting the data and then identifying behavior patterns that could be used for hypothesis creation. However, this exploratory approach would increase the size of the analysis task considerably, since the number of coding categories so derived would tend to be high and the identification of behavior patterns is normally a complex activity.

Conclusions

This article has described a methodology for the capture and analysis of hybrid data. The specific area of application for this methodology is computerized tasks in which the user has to interpret and coordinate multiple representations presented on the computer screen. This methodology is explained by employing an example from software troubleshooting, but the methodology should also be applicable to other troubleshooting activities.

We achieved the capture of data through the SDE, a computerized environment that employs a restricted focus technology that enables researchers to track the user's visual attention by blurring the stimuli presented on the screen and allowing the participant to see only a small region of the stimulus in focus at any one time. This environment records what the participant is focusing on at a point in time, thus enabling the capture of the moment-by-moment focus of visual attention. In addition, it records user-computer keyboard interaction and what participants say.

The analysis of hybrid data is performed by building an interpretation of the participant's behaviors, taking into account all of the available types of data. This interpretation classifies the participant's behaviors into predefined categories that are associated with the set of hypotheses to be explored. The occurrence frequency of these behaviors can be tested quantitatively to prove (or disprove) the proposed hypotheses.

Finally, this article discusses some limitations of the approach that have to do with both the capture and analysis of hybrid data.

AUTHOR NOTE

This work was supported by EPSRC Grant GR/N64199 and Nuffield Foundation Grant URB/01703/G. Support for Richard Cox from the Leverhulme Foundation (Leverhulme Trust Fellowship G/2/RFG/2001/0117) and the British Academy is gratefully acknowledged. The authors thank Stephen Grant for refining the coding categories and performing the coding tasks. Correspondence relating to this article may be sent to P. Romero, Human Centred Technology Group, Informatics Department, University of Sussex, Brighton BN1 9QH, England (e-mail: pabl@sussex.ac.uk).

REFERENCES

- AINSWORTH, S. (1999). The functions of multiple representations. *Computers & Education*, **33**, 131-152.
- BEDNARIK, R., & TUKIAINEN, M. (2004). Visual attention and representation switching in Java program debugging: A study using eye movement tracking. In E. Dunican & T. R. G. Green (Eds.), *Proceedings of the 16th Annual Workshop of the Psychology of Programming*

- Interest Group* (pp. 159-169). Available at www.ppig.org/workshops/16th-programme.html.
- BERGANTZ, D., & HASSELL, J. (1991). Information relationships in PROLOG programs: How do programmers comprehend functionality? *International Journal of Man-Machine Studies*, **35**, 313-328.
- BLACKWELL, A. F., JANSEN, A. R., & MARRIOTT, K. (2000). Restricted Focus Viewer: A tool for tracking visual attention. In M. Anderson, P. Cheng, & V. Haarslev (Eds.), *Theory and application of diagrams: First International Conference, Diagrams 2000* (pp. 162-177). Berlin: Springer.
- CHI, M. T. H. (1997). Quantifying qualitative analyses of verbal data: A practical guide. *Journal of the Learning Sciences*, **6**, 271-315.
- COX, R. (1997). Representation interpretation versus representation construction: A controlled study using switchERII. In B. du Boulay & R. Mizoguchi (Eds.), *Artificial intelligence in education: Knowledge and media in learning systems. Proceedings of the 8th World Conference of the Artificial Intelligence in Education Society* (pp. 434-441). Amsterdam: IOS.
- COX, R., & LUM, C. (2004). Case-based teaching and clinical reasoning: Seeing how students think with PATSy. In S. Brumfitt (Ed.), *Innovations in professional education for speech and language therapists*. London: Whurr.
- COX, R., O'DONNELL, M., & OBERLANDER, J. (1999). Dynamic versus static hypermedia in museum education: An evaluation of ILEX, the intelligent labelling explorer. In S. P. Lajoie & M. Vivet (Eds.), *Proceedings of the 9th Artificial Intelligence in Education (AI-ED99) Conference, Le Mans, France, July, 1999* (pp. 181-188). Amsterdam: IOS.
- CROSBY, M., & STELOVSKY, J. (1989). Subject differences in the reading of computer algorithms. In G. Salvendy & M. J. Smith (Eds.), *Designing and using human-computer interfaces and knowledge based systems* (pp. 137-144). Amsterdam: Elsevier.
- DAVIES, S. P. (1994). Knowledge restructuring and the acquisition of programming expertise. *International Journal of Human-Computer Studies*, **40**, 703-726.
- DE JONG, T., AINSWORTH, S., DOBSON, M., VAN DER HULST, A., LEVONEN, J., REIMANN, P., ET AL. (1998). Acquiring knowledge in science and mathematics: The use of multiple representations in technology-based learning environments. In M. W. van Someren, P. Reimann, H. P. A. Boshuizen, & T. de Jong (Eds.), *Learning with multiple representations* (pp. 9-40). Amsterdam: Pergamon.
- DENZIN, N. K. (1997). Triangulation in educational research. In J. P. Keeves (Ed.), *Educational research, methodology, and measurement: An international handbook* (2nd ed., pp. 318-322). New York: Pergamon.
- DÉTIENNE, F. (1997). Assessing the cognitive consequences of the object-oriented approach: A survey of empirical research on object-oriented design by individuals and teams. *Interacting With Computers*, **9**, 47-72.
- DU BOULAY, B., ROMERO, P., COX, R., & LUTZ, R. (2003). Towards a debugging tutor for object-oriented environments. In V. Alevan, U. Hoppe, J. Kay, R. Mizoguchi, H. Pain, F. Verdejo, & K. Yacef (Eds.), *Supplementary Proceedings of Artificial Intelligence in Education Conference (AIED2003), Sydney, Australia* (pp. 399-407). Sydney: University of Sydney.
- FRIEDRICH, G., STUMPTNER, M., & WOTAWA, F. (1999). Model-based diagnosis of hardware designs. *Artificial Intelligence*, **111**, 3-39.
- GABBAY, F., & MENDELSON, A. (1999). The "smart" simulation environment—A tool-set to develop new cache coherency protocols. *Journal of Systems Architecture*, **45**, 619-632.
- GILMORE, D. J. (1991). Models of debugging. *Acta Psychologica*, **78**, 151-172.
- GILMORE, D. J., & GREEN, T. R. G. (1988). Programming plans and programming expertise. *Quarterly Journal of Experimental Psychology*, **40A**, 423-442.
- HOUNTALAS, D. T., & KOUREMENOS, A. D. (1999). Development and application of a fully automatic troubleshooting method for large marine diesel engines. *Applied Thermal Engineering*, **19**, 299-324.
- JANSEN, A. R., BLACKWELL, A. F., & MARRIOTT, K. (2003). A tool for tracking visual attention: The Restricted Focus Viewer. *Behavior Research Methods, Instruments, & Computers*, **35**, 57-69.
- KRANZLMÜLLER, D., GRABNER, S., & VOLKERT, J. (1997). Debugging with the MAD environment. *Parallel Computing*, **23**, 199-217.
- MARZI, R., & JOHN, P. (2002). Supporting fault diagnosis through a multi-agent architecture. *Journal of Engineering Manufacture*, **216**, 627-631.
- MULHOLLAND, P. (1997). Using a fine-grained comparative evaluation technique to understand and design software visualization tools. In S. Wiedenbeck & J. Scholtz (Eds.), *Empirical Studies of Programmers, Seventh Workshop* (pp. 91-108). New York: ACM.
- PAPADOPOULOS, Y. (2003). Model-based system monitoring and diagnosis of failures using state charts and fault trees. *Reliability Engineering & System Safety*, **81**, 325-341.
- PATEL, M. J., DU BOULAY, B., & TAYLOR, C. (1997). Comparison of contrasting Prolog trace output formats. *International Journal of Human-Computer Studies*, **47**, 289-322.
- PENNINGTON, N. (1987). Comprehension strategies in programming. In G. M. Olson, S. Sheppard, & E. Soloway (Eds.), *Empirical Studies of Programmers, Second Workshop* (pp. 100-113). Norwood, NJ: Ablex.
- PENNINGTON, N., LEE, A. Y., & REHDER, B. (1995). Cognitive activities and levels of abstraction in procedural and object-oriented design. *Human-Computer Interaction*, **10**, 171-226.
- ROBERTSON, S. P., DAVIS, E. F., OKABE, K., & FITZ-RANDOLF, D. (1990). Program comprehension beyond the line. In D. Diaper, D. J. Gilmore, G. Cockton, & B. Shackel (Eds.), *INTERACT '90: Proceedings of the 3rd IFIP International Conference on Human-Computer Interaction* (pp. 959-963). Amsterdam: North-Holland.
- ROMERO, P., COX, R., DU BOULAY, B., & LUTZ, R. (2002). Visual attention and representation switching during Java program debugging: A study using the Restricted Focus Viewer. In M. Hegarty, B. Meyer, & N. H. Narayanan (Eds.), *Diagrammatic representation and inference: Second International Conference, Diagrams 2002* (pp. 221-235). Berlin: Springer.
- ROMERO, P., DU BOULAY, B., LUTZ, R., & COX, R. (2003). The effects of graphical and textual visualisations in multi-representational debugging environments. In J. Hosking & P. Cox (Eds.), *Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments* (pp. 236-238). Auckland, New Zealand: IEEE Computer Society.
- ROMERO, P., LUTZ, R., COX, R., & DU BOULAY, B. (2002). Co-ordination of multiple external representations during Java program debugging. In S. Wiedenbeck & M. Petre (Eds.), *Proceedings of the 2002 IEEE Symposia on Human Centric Computing Languages and Environments* (pp. 207-214). Arlington, VA: IEEE Press.
- STENNING, K., COX, R., & OBERLANDER, J. (1995). Contrasting the cognitive effects of graphical and sentential logic teaching: Reasoning, representation and individual differences. *Language & Cognitive Processes*, **10**, 333-354.
- VESSEY, I. (1985). Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies*, **23**, 459-494.
- VESSEY, I. (1989). Toward a theory of computer program bugs: An empirical test. *International Journal of Man-Machine Studies*, **30**, 23-46.

NOTES

1. The SDE source code can be downloaded from www.informatics.sussex.ac.uk/projects/crusade.
2. A Quicktime movie file containing a fraction of a debugging session from these studies can be found at www.informatics.sussex.ac.uk/projects/crusade/clips/subj26.mov.

(Manuscript received February 6, 2006;
accepted for publication February 10, 2006.)