Production Number B204

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
Informatics Department
Sussex University, U.K.
email: pablor@sussex.ac.uk

Abstract

This paper describes a methodology for the capture and analysis of hybrid data. A case study in the field of reasoning with multiple representations and 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 through 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 analysed via a methodology which combines qualitative and quantitative approaches. The paper describes the software tool employed and the analytic methodology and also discusses data capture issues and limitations of the approach.

Keywords: rich data analysis, external representations

1 Introduction

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

Throughout this paper, we refer to this mixture of data of different types but which originate from the same empirical episode as *hybrid data*. These are rich data which 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 a single data type approach (Denzin, 1997). Computerised tasks offer a good opportunity to capture hybrid data as the user-computer interaction can be logged, verbalisations can be digitally recorded and performance data can also be registered.

We are particularly interested in computerised 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 computerised activity performed using a graphical user interface involves working with multiple windows. Specifically our focus is on understanding representation coordination in reasoning and learning (de Jon, Ainsworth, Dobson, van der Hulst, Levonen & Reimann, 1998; Ainsworth, 1999) and troubleshooting-based problem solving. Troubleshooting within a computerised environment is an activity particularly suitable for studying representation coordination because these sorts of environment normally provide concurrently displayed, adjacent, dynamic, multiple and linked representations. Computerised environments for troubleshooting normally simulate (and enable users to monitor) the system under inspection by offering a set of visualisations which change over time and which give a detailed account of the behaviour 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 the faulty behaviour.

Performing troubleshooting activities within a computerised environment is a task which is particularly suitable for capturing process data. This sort of data can include a record of i) the interaction events the user performed in order to control the execution of the simulation, ii) the windows and visualisations employed and iii) the user's verbalisations, among other information. Computerised troubleshooting environments can be complemented with the appropriate functionality to capture these sorts of data.

In this paper we describe a methodology for the capture, analysis and synthesis of hybrid

data focusing on the coordination of multiple representations when working with computerised 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 is particularly relevant to the study of program debugging as programmers normally work within a computerised 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 the program code, the output of the program and a set of dynamic visualisations that illustrate the state of the program at different moments of its execution. Studying novices' debugging behaviour offers useful information about the development of programming skill, common misconceptions and efficient debugging strategies.

This paper comprises 6 sections. Section 2 presents a description and analysis of the troubleshooting activity and the role of external representation within it. Section 3 presents 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. The subsequent three sections, 4, 5, and 6 describe a methodology for data capture and analysis. Finally Section 7 presents a discussion and critique of the approach and suggests possible lines of further research.

2 Representation coordination in troubleshooting tasks

In this paper we describe an approach to hybrid data capture and analysis which 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 multi-representational system. Examples of this situation are found in applications as diverse as fault finding in industrial machine-tools (Kranzlmuller, Grabner & Volkert, 1997), in parallel and distributed computer software (Marzi & John, 2002; Gabbay & Mendelson, 1999), 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 is illustrated in more detail with a concrete example in the area of commercial call centre systems. In this example, when errors are reported the inspection and coordination of a number of log files is needed in order to ascertain the nature of the problem and the subsystem in which it occurred. In commercial call centre systems, log files in different formats are sometimes produced by three different subsystems: the telephony, the user interface and the software connecting these two.

The telephony subsystem produces a log file with time-stamped entries indicating changes in telephony state. For example, showing when a telephone was in use, or free to take a call, or blocking calls. The user interface subsystem produces another log file which indicates which key presses had taken place in the telephony window which was used to control the telephone system. These two representations allow the programmer to ascertain whether the computer telephony integration system was 'in sync' with the telephone. If this was not the case, some remedial action would be required. The third representation is the trace from the software code. This shows in detail which methods have been 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 multi-faceted 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.

3 Capturing and analysing 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 analyse performance rather than process data (Vessey, 1989; Gilmore & Green, 1988; Gilmore,

1991; Davies, 1994; Patel, du Boulay & Taylor, 1997). These sorts of study have usually captured information about percentages of correct responses and solution times for given debugging tasks. However when they have considered process data, the most popular types have been verbalisations (Vessey, 1985; Pennington, 1987; Pennington, Lee & Rehder, 1995; Mulholland, 1997). Experimental participants have been asked to 'think-aloud' and their verbal protocols have been analysed to explore strategy and to investigate how it relates to programming experience and proficiency (Vessey, 1985); to explore the relationships between notational properties of the language, the computerised debugging environment and the information types programmers consider as 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 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 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 eye-trackers (Crosby & Stelovsky, 1989). Such studies have analysed code reading patterns to investigate whether they are more similar to prose reading or to tasks related to problem solving (Robertson, Davis, Okabe & Fitz-Randolf, 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 analyse 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 evaluating virtual museum applications (Cox, O'Donnell & Oberlander, 1999), investigating individual differences in logic proof development (Stenning, Cox & Oberlander, 1995) and for studying health science students' diagnostic reasoning skill acquisition (Cox & Lum, 2004) among other areas. Because of the characteristics mentioned above, we believe that

hybrid data capture and analysis is a suitable methodology for a deep study of troubleshooting activities when performed in a computerised environment. The following section describes the computerised environment which, complemented with the appropriate functionality, has been employed for hybrid data capture in troubleshooting tasks.

4 RFV technology

The Restricted Focus Viewer (RFV) is a program which 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 a 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; enabling in this way the capture of moment-by-moment focus of visual attention. As the RFV can present several visual stimuli on the computer screen at the same time, it enables the user's representation switching between concurrently displayed, adjacent representations to be captured for later analysis.

The user interaction data captured by the RFV can be read in by Replayer, a companion program that can replay the way that the RFV participant moved the focus window over the stimuli. This companion program can be used as an analysis tool to replay experimental sessions allowing the researcher to pause, re-start or play back the locus of movement of the unblurred region faster or slower than real-time among other functions.

The RFV records changes in the location of the unblurred region with millisecond precision. The data captured by this program indicates, 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 to the nature of the task and to the nature 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, Jansen & Marriott, 2000). That validation study found no significant differences in the inspection strategies of participants working with this

technology and with eye-tracking 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 Section 7.

5 From single type to hybrid data capture

<< INSERT FIGURE 1 HERE >>

Recording and analysing data about focus of visual attention can offer interesting information about the process of, in this case, coordinating multiple external representations. However, troubleshooting environments, besides presenting visualisations of the system under inspection, allow users to view the execution of simulations for specific cases. The SDE (Software Debugging Environment), is a modified version of the RFV which incorporates functionality to enable users to view snapshots of the pre-computed execution for 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 horizontal line 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), output of the program (bottom right), and subroutine call stack (bottom centre); 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' verbalisations [1].

5.1 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, Lutz, Cox & du Boulay, 2002; Romero, du Boulay, Lutz & Cox, 2003). However, the fact that the display of simulations in the SDE is achieved through pre-constructed 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 visualisations depicting states of the system under study at different moments of the

simulation for a specific example. The SDE can present this sequence in steps. The user can move between the sequence by pressing the arrow keys, can select to see the visualisations associated with the beginning of the execution via the <home> key and those associated with the end of it via 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 visualisations depicting states of the system at different moments of the simulation, remains unchanged. The layout and number of windows displayed are parameters set in an initialisation 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, as moving the unblurred spot is done with the mouse, any manipulation to 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 website navigation given that the main form of interaction in this context is clicking on hyper-links situated inside the displayed windows.

Presenting visualisations of the state of a system as pre-computed image stimuli brings additional requirements to the SDE. First, these visualisations 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 which is organised 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 will encourage participants to switch their focus of visual attention between these representations. This feature will then produce two different forms of mouse usage. The first is the original one, moving the mouse to change the unblurred spot while inspecting a specific window. The second is moving the mouse to switch between windows. In order to differentiate between these two forms of mouse usage 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 feature makes switching between windows easier, because participants do not have to re-establish the place where they were looking at every time they switch their attention from one window image back to an earlier one (Romero, Cox, du Boulay & Lutz, 2002). Also, this change enabled us to record (and analyse) window switching and stimulus inspection as two different behaviours.

5.2 Capturing data of different types

Besides capturing data about the focus of visual attention (mouse movements and clicks), the SDE also records information about the control of the view of the simulated execution and the participants' verbalisations. The keyboard actions that participants perform to control the presentation of the simulation execution (arrows, <home> and <end> key pressing) are recorded in the log file that this application generates. In this way, the Replayer program is able to replay, besides 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', this audio data can be recorded by the SDE. The participants' verbalisations are digitally recorded onto the computer's hard disk in Unix audio-file format (*.au*, compatible with most *pc* and *macintosh* audio applications) with a 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 analysed in a synchronous way. This hybrid data capture has been employed in the studies reported in Romero, Cox, du Boulay & Lutz (2002), Romero, Lutz, Cox & du Boulay (2002) and Romero, du Boulay, Lutz & Cox (2003) [2].

6 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 analyses these data both quantitatively and with a methodology that combines quantitative and qualitative approaches. The quantitative analysis relates participants' performance to SDE usage while the combined approach explores participants' behaviour and strategy by taking into account three types of data synchronically: focus of attention trace, control of the presentation of the simulation's execution and verbalisation data.

6.1 Quantitative analysis

The quantitative analysis requires participants' performance to be extracted from the verbalisation data and the log file (of keyboard and mouse actions) to be transformed into a description of simulation execution and window fixations and switches. The latter task can be automated (through a computer program that performs this transformation) but the analysis of the verbalisation data needs to be performed by human raters.

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

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

The content and format of the visualisations presented by the environment can be manipulated. For example, the version of the SDE employed for the studies on representation coordination in debugging reported in Romero, Cox, du Boulay & Lutz

(2002), Romero, Lutz, Cox & du Boulay (2002) and Romero, du Boulay, Lutz & Cox (2003) presents either graphical or textual visualisations that highlight 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 analysed looking for significant main and interaction effects.

Finding out about these relationships offers important information about patterns of environment usage and programming expertise. This information can be complemented by investigating the debugging strategies that shaped this environment usage.

6.2 Combining quantitative and qualitative methods

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

This 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 (like the one described in Section 6.1, for example). The main idea behind our proposed analysis methodology is to build an interpretation of the participant's behaviour by taking into account all the available types of data. This interpretation classifies the participant's behaviour into predefined categories associated with the set of hypotheses to be explored. The occurrence frequency of these behaviours can be tested quantitatively to provide evidence for (or against) the proposed hypotheses.

The qualitative analysis method 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' verbalisations. As 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 is for the event number, the second one is for the programmer's utterances, and the third, fourth, fifth and sixth are for unblurred information displayed by the different windows of the SDE (the code, objects, call sequence

and output windows as shown in Figure 1). Only the contents of the currently unblurred window are shown in the chart. Each row of this table codes one debugging event. In general, debugging events are bounded by pauses or changes of topic in programmers' verbalisations (utterances), inter-window switches of visual attention focus or breakpoint switches. The unit of verbalisation that we considered as an utterance is that verbalisation which is limited by a considerable pause (eg. a pause greater than 2 minutes) and/or by a change of topic. Inter-window switches occur when the user moves the unblurred area from one window to another and breakpoint switches take place when the programmer moves 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).

<< INSERT TABLE 1 HERE >>

A basic hypothesis that can be considered to explain and exemplify the classification of behaviours into different categories has to do with the possible relationship between debugging performance and behaviour. Such a hypothesis suggests that certain program comprehension and debugging strategies are more effective than others. In order to test this hypothesis, a set of coding categories defining different comprehension and debugging strategies has to be defined and the participant behaviours recorded in the coding sheets have to be classified according to these coding categories.

6.3 Program comprehension and debugging behaviour coding categories

In order to code this data, a coding protocol had to be developed and operationalised (Chi, 1997). The development of this coding protocol required decisions regarding the number and type of categories to include and the 'grain size' of programming event that the coding categories represented. These decisions in turn depended on the hypothesis being tested, the task and the content domain. The operationalisation of the coding protocol required to verify that there was a clear relationship between behaviours 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

operationalisation phases can be considered as top-down and bottom-up processes which are normally applied in an iterative fashion when refining the coding protocol (Chi, 1997).

<< INSERT TABLE 2 HERE >>

A partial list of behaviour coding categories is presented in Table 2. This table is divided into program comprehension and program debugging coding categories. According to research in program comprehension and debugging (Vessey, 1985; Pennington, 1987; Davies, 1994; Pennington, Lee & Rehder, 1995; Détienne, 1997), some of these behaviours are more successful than others. For example, coding category *C8* is an instance of a successful program comprehension coding category because it is intended to register the occurrence of a strategy that tries to link the program domain and the problem domain. Such a cross-referencing behaviour is associated with good debugging performance (Pennington, 1987). On the other hand program debugging coding category *D13* is intended to register the occurrence of a strategy which, for example, tries to understand specific details of the program without having a clear idea of what the effects of the error are in terms of the output. Such early preoccupation with program detail is associated with poor debugging performance.

6.4 Data encoding

To obtain a detailed characterisation of programmers' program comprehension and debugging strategies, their coding sheets are analysed to look for occurrences of the behaviour coding categories in Table 2. This is a procedure that has to be performed by a human rater and requires her/him to consider information about focus of attention, program execution and programmers' verbalisations synchronously. For example, coding category *C7* requires verification of whether simple stepping (a program execution behaviour) happened while the focus of visual attention was located in a specific window (the objects view). Verbalisations referring to executing the program in steps and/or to the contents of the objects window can strengthen the case for acknowledging the occurrence of this behaviour. Table 3 presents an example of data encoding for a segment of a debugging session (the coding sheet in Table 1 refers to the same debugging session segment). In this table, events 9 to 12 have been categorised as occurrences of the behaviour 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 comprises information about transitions that bounded debugging events. As mentioned in Section 6.2, 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).

<< INSERT TABLE 3 HERE >>

These low level behaviour coding categories can be integrated into program comprehension and debugging episodes (Vessey, 1985). These episodes are groups of behaviours which have a specific goal in common and which can be used to identify programmers' strategies.

A cluster analysis can allow us to categorise groups of programmers according to their displayed strategies and to compare this categorisation with their performance data. This categorisation can also be complemented with the findings of the quantitative analysis. In this way, a model of program comprehension and debugging expertise in terms of behaviour and strategy can be empirically derived.

This way of deriving a program comprehension and debugging model, taking into account several types of data synchronously, has advantages over methods that have considered only one type of information. First, the range of behaviours, and therefore strategies, taken into account in the model can be wider. For example, behaviours *C4*, *C11* and *C19* would be difficult to take into account in a model that only considers verbal data. Also, having several types of data enables the encoding process to have a higher level of certainty (as in the example about coding category *C7* above).

Considering a wide range of strategies in a program comprehension and debugging model might 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 that have to do with focus of visual attention can enable the environment, in principle, to give advice on these matters. The learning environment could, for example, embody a number of monitoring rules that kept dynamic track of both focus of attention and switching behaviour to guide the student to pay attention in more sensible places.

7 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 is divided into problems with data capture and with data analysis.

Issues related to data capture involve assumptions relating focus of visual attention and 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 eye-tracking equipment (Blackwell, Jansen & Marriott, 2000; Romero, Cox, du Boulay & Lutz, 2002; Jansen, Blackwell & Marriott, 2003). However, similar validation studies for the SDE (Bednarik & Tukiainen, 2004) employing both the restricted focus technology and an eye-tracker have found that participants glance at blurred areas with a certain frequency. This behaviour could be caused by saccades or could be part of a strategy to minimise 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 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, eye-tracking technology. According to the SDE validation study by Jansen, Blackwell & Marriott (2003), participants using eye-tracking do more representation switching than those employing the restricted focus technology. This difference was not relevant to our studies (Romero, Cox, du Boulay & Lutz, 2002; Romero, Lutz, Cox & du Boulay, 2002; Romero, du Boulay, Lutz & Cox, 2003), as the general pattern of interaction and relative number of switches between representations do not seem to be significantly modified. However, this difference in the level of granularity of 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 visualisations when performing representation switching. The fact that each window 'remembers' its region in focus has made window switching easier for participants. Indeed, in the SDE validation study reported in Romero, Cox, du Boulay & Lutz (2002), there was a tendency for participants to perform better when working with the restricted focus environment than when working with the one without a restricted view. One of the possible causes for this tendency is this modification in the fixation and switching tasks. Unfortunately taking the 'bookmark' feature away seemed to increase the difficulty of the task considerably as re-positioning 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, du Boulay & Lutz (2002) was not significant but only a tendency, it was assumed that the modification of the fixation and switching tasks does 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 pre-defined set of hypotheses.

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

There is also a need for a pre-defined set of hypotheses. Defining these hypotheses is necessary for the specification of the behaviour coding categories. An alternative is to perform this specification in a bottom-up fashion, through inspecting the data and then identifying behaviour patterns which could be used for hypotheses creation. However, this exploratory approach increases the size of the analysis task considerably as the number of coding categories so derived tends to be high and the identification of behaviour patterns is normally a complex activity.

8 Conclusions

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

The capture of data is achieved through the SDE, a computerised environment that employs a restricted focus technology which 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 moment-by-moment focus of visual attention. Additionally, 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 behaviour (taking into account the data of all the available types). This interpretation classifies the participant's behaviour into predefined categories which are associated with the set of hypotheses to be explored. The occurrence frequency of these behaviours can be tested quantitatively to prove (or disprove) the proposed hypotheses.

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

Acknowledgments

This work was supported by the EPSRC grant GR/N64199 and the Nuffield Foundation grant URB/01703/G. The support for Richard Cox of the Leverhulme Foundation (Leverhulme Trust Fellowship G/2/RFG/2001/0117) and the British Academy is gratefully acknowledged. The authors wish to thank Stephen Grant for his hard work both in refining the coding categories and in the coding tasks.

References

Ainsworth, S. (1999). The functions of multiple representations. *Computers & Education*, *33*(2-3), 131–152.

Bednarik, R. & Tukiainen, M. (2004). Visual attention and representation switching in Java program debugging: a study using eye-movement tracking. In Dunican, E. & Green, T. (Eds.), *Proceedings of the 16th annual workshop of the Psychology of Programming Interest Group*, (pp. 159–169).

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., Jansen, A., & 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. Lecture Notes in Artificial Intelligence 1889* (pp. 162–177). Springer-Verlag.

Chi, M. T. H. (1997). Quantifying qualitative analyses of verbal data: a practical guide. *The Journal of the Learning Sciences*, *6*(3), 271–315.

Cox, R. (1997). Representation interpretation versus representation construction: a controlled study using switcherii. In du Boulay, B. & Mizoguchi, R. (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. 434–444)., 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*. 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, The Netherlands: IOS Press.

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, The Netherlands: Elsevier science publishers B.V.

Davies, S. P. (1994). Knowledge restructuring and the acquisition of programming expertise. *International Journal of Human Computer Studies*, *40*, 703–726.

de Jon, T., Ainsworth, S., Dobson, M., van der Hulst, A., Levonen, J., & Reimann, P. (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 Jon (Eds.), *Learning with Multiple Representations* (pp. 9–40). Oxford, U.K.: Elsevier Science.

Denzin, N. K. (1997). Triangulation in educational research. In J. P. Keeves (Ed.), *Educational research, methodology and measurement: an international handbook* (pp. 318–322). Oxford, UK: Elsevier Science Ltd.

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 Aleven, V., Hoppe, U., Kay, J., Mizoguchi, R., Pain, H., Verdejo, F., & Yacef, K. (Eds.), *Supplementary Proceedings of Artificial Intelligence in Education Conference (AIED2003), Sydney, Australia*, (pp. 399–407). 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*(1), 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. & Kouremenos, A. (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*(4), 57–69.

Kranzlmuller, 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. *Mathematics and Computers in Simulation*, *60*, 217–224.

Mulholland, P. (1997). Using a fine-grained comparative evaluation technique to understand and design software visualization tools. In Wiedenbeck, S. & Scholtz, J. (Eds.), *Empirical Studies of Programmers, seventh workshop*, (pp. 91–108)., New York. ACM press.

Papadopoulos, Y. (2003). Model-based system monitoring and diagnosis of failures using statecharts and fault trees. *Reliability Engineering and System Safety 81 (2003) 325341*, *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 Olson, G. M., Sheppard, S., & Soloway, E. (Eds.), *Empirical Studies of Programmers, second workshop*, (pp. 100–113)., Norwood, New Jersey. 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 Diaper, D., More, D., Cockton, G., & Shackel, B. (Eds.), *Human-Computer Interaction - INTERACT '90*, (pp. 959–963).

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 Hegarty, M., Meyer, B., & Narayanan, N. H. (Eds.), *Diagrammatic Representation and Inference. Second International Conference, Diagrams 2002. Lecture Notes in Artificial Intelligence 2317*, (pp. 221–235).

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.), *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.), *2002 IEEE Symposia on Human Centric Computing Languages and Environments* (pp. 207–214). Airlington, Virginia, USA: 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 and Cognitive Processes*, *10*(3/4), 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*(1), 23–46.

Notes

[1]The SDE source code can be downloaded from *http://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 *http://www.informatics.sussex.ac.uk/projects/crusade/clips/subj26.mov*

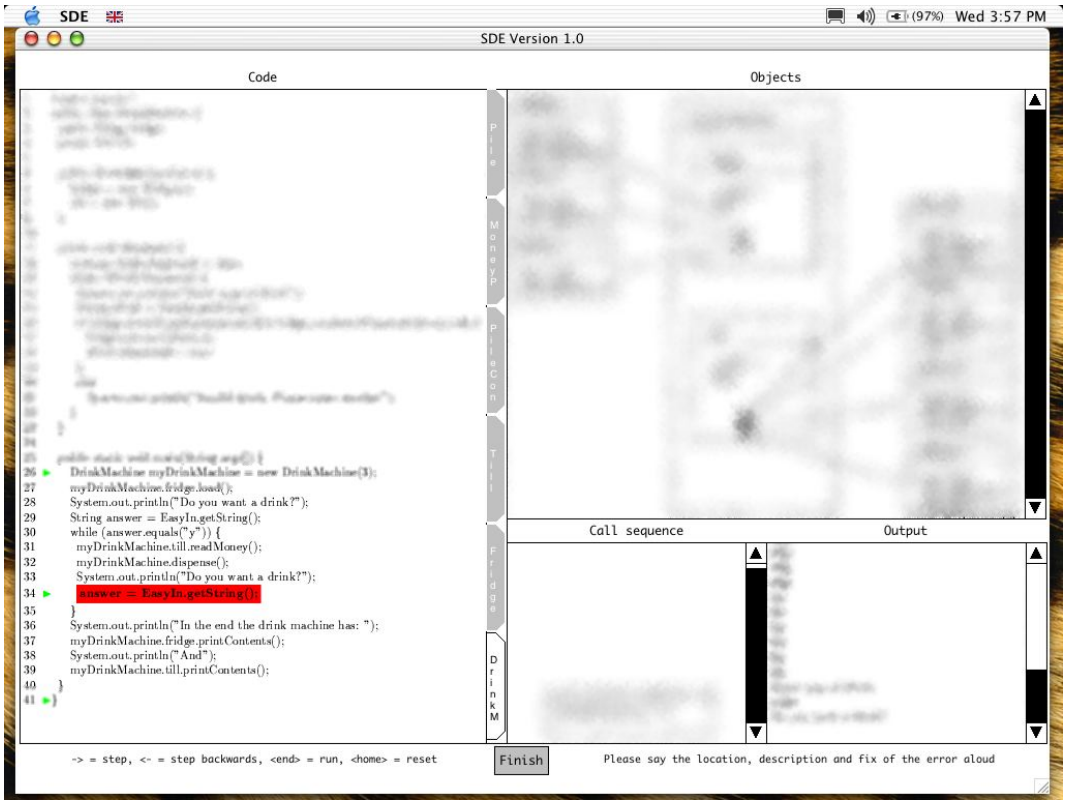

Figure 1: The SDE employed for rich data capturing. The unblurred region is located towards the bottom of the leftmost window

Event number	**Verbalisation**	**Focus of attention**			
		Code	**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]		

Table 1: Section of coding sheet for a specific debugging session

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 watching the objects view
C8	Whilst looking at code or object view, utterances reflect *real world* objects in the problem domain
C9	Looking at the output or object view whilst talking about the code
C10	Utterances relating to higher level entities (e.g. method, sub-routine, 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 verbalisation
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 watching the objects view
D2	Considering negative evidence in their reasoning
D3	Focusing in on an area of code after an uttered hypothesis
D4	Re-running the code with fix in place
D5	Temporarily considering regions of the program as free of errors
D6	Attempting an a priory 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 an 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	Talk in terms of breakpoints (dynamic view) but not stepping through
D12	Utterances of code cliches
D13	Early delving into the details

Table 2: Sample comprehension and debugging behaviour coding categories

Event number	**behaviour category**	**Transition**
8	C22	pause
9	C7	switch/pause
10	C7	switch/pause
11	C7	switch
12	C7, C4	switch/pause

Table 3: Section of data coding for a specific debugging session