

Coordination of Multiple External Representations in Learning Programming: Review Report GR/N64199/01

Background/Context

Only a limited number of studies have looked at the issue of representation coordination during program comprehension and debugging in multi-representational programming environments and at how various factors determine the behaviour and strategies of programmers in this context. These factors include the programmers' level of experience and their cognitive individual differences as well as the form and content of the representations available.

One approach to study representation coordination is to analyse the focus of programmers' visual attention while performing programming tasks. Studies analysing visual attention focus have normally concentrated on one representation, the program code (Widowsky & Eyerth, 1986; Crosby & Stelovsky, 1989). Interesting questions concerning focus of visual attention are: i) to what extent do programmers employ each of the representations provided and ii) are there patterns of representation use which characterise better programming performance.

Studies in other areas have suggested that representation coordination is not an easy task and that its success depends, among other factors, on individual ability (Ainsworth, Wood, & Bibby, 1996). One important question in this context is what makes some people good at working with multi-representational environments.

Two important aspects to consider regarding the coordination of multiple representations in programming are modality and perspective (de Jon, Ainsworth, Dobson, van der Hulst, Levonen, & Reimann, 1998). The term 'modality' is used here to mean the representational forms used to present or display information, rather than in the psychological sense of sensory channel. A typical modality distinction is between propositional and diagrammatic representations. Thus, this first aspect refers to coordinating representations which are basically propositional with those that are mainly diagrammatic. It is not clear whether coordinating representations with the same modality type has advantages over working with mixed multiple representations or whether including a high degree of graphicality has potential benefits for performing the task (Ainsworth et al., 1996).

While modality is concerned with form, perspective is concerned with content. Perspective refers to the programming information types that a representation highlights. Computer programs are information structures that comprise different types of information (Pennington, 1987), and programming notations usually highlight some of these aspects at the cost of obscuring others (the *match-mismatch hypothesis*, (Gilmore & Green, 1984)). Some of these different information types are: function, data structure, operations, data-flow and control-flow. Research in program comprehension for Object-Oriented languages suggests that these kinds of language highlight function as well as static data element information whilst obscuring control-flow information (Corritore & Wiedenbeck, 1999; Wiedenbeck & Ramalingam, 1999). However, it is not clear whether novice programmers working with medium size programs find comprehending function and static data element information in Object-Oriented languages an easy task (Wiedenbeck, Ramalingam, Sarasamma, & Corritore, 1999), especially because as program size increases, this sort of information tends to become diffuse.

Key Advances and Supporting Methodology

The key advances of this work have been

- An identification and analysis of the representation formalisms commonly employed to visualise the execution of Object-Oriented programs by a range of programming tools and environments (Romero, Cox, du Boulay, & Lutz, 2003a).
- The development of an experimental tool that enables studies of program debugging to be undertaken and logs visual attention fixations and switches as well as verbal protocols. The tool allows sessions to be replayed later in real time (Romero, du Boulay, Cox, & Lutz, 2003b).
- The detailed analyses of debugging behaviour in two experiments that logged representation use and related this to a range of individual difference variables as well as to notational properties of the representations that have to do with their form and content (Romero, Lutz, Cox, & du Boulay, 2002; Romero, du Boulay, Lutz, & Cox, 2003c).
- The development of a set of design principles for program comprehension aids, and particularly for learning environments for Java program comprehension and debugging (du Boulay, Romero, Cox, & Lutz, 2003).

Project Plan Review

Representation formalisms employed in Object-Oriented tools

The representation formalism employed is one of the most important aspects of the program visualisations provided by comprehension and debugging tools. Part of the design of the SDE involved defining these sorts of representation formalisms. Given that there are already a variety of commercial software development environments, the first step in defining our own representations was to identify and analyse the representation formalisms employed by some of the most popular Object-Oriented programming environments to support tasks involving program comprehension. These representations were compared in terms of the programming aspects they highlight and of their information modality.

The results of this survey indicated that two important information types for these software development environments are data structure and control-flow, and that the representations employed by them are mostly textual. Common characteristics in the representation formalisms of these programming environments were identified. An analysis of these common characteristics in terms of Green's (1989) Cognitive Dimensions revealed that two potential problems can be difficulty in coordinating the different representations and the cognitive load that they might impose when used to communicate dynamic aspects of the program execution (for details see Romero et al. (2003a)).

A computerised experimental tool

In order to log data from users, including their focus of attention, a software development environment SDE was implemented on top of a modified version of the Restricted Focus Viewer (RFV) (Blackwell, Jansen, & Marriott, 2000). The SDE presents image stimuli in a blurred form. When the user clicks an image, a section of it around the mouse pointer becomes focused. In this way, the program restricts how much of a stimulus can be seen clearly and allows visual attention to be tracked as the user moves an unblurred 'foveal' area around the screen. Use of the SDE enabled moment by moment representation switching between concurrently displayed, adjacent representations to be captured for later analysis.

Our Java SDE enabled participants to see the program code, its output for a sample execution, and various visualisations such as data structure and control-flow snapshots at particular breakpoints in the

execution of the program. We demonstrated that visual attention tracking methods, and more specifically a tool like the Restricted Focus Viewer (RFV) could be used to investigate issues related to the process of coordinating multiple external representations in program debugging (Romero, Cox, du Boulay, & Lutz, 2002). The investigation suggested that debugging performance was not affected by this method of tracking visual attention and that there might be fixation and switching patterns characteristic of superior debugging in this context. The Java SDE built is available as an open source application at the project's website (www.cogs.susx.ac.uk/projects/crusade).

Coordination of multiple external representations

Investigating the coordination of multi-representational environments for novice Java programmers has provided findings that can be categorised into four interrelated areas: a) representation use, b) individual differences, c) modality and d) perspective.

The following sections conflate the results of the two empirical studies carried out; one with a static SDE and the other with a dynamic SDE (as specified in the project plan).

Representation use. Although the code is the representation most frequently used by novice programmers, programming experience is related to a balanced use of the available representations. While less experienced programmers show a low degree of interaction with the support visualisations, more experienced ones interact more with these representations (for details see Romero et al. (2002)).

An important aspect of representation use is the frequency of visual attention switching between the different representations in the environment. According to Vessey (1985), debugging ability is inversely related to switches of focus between the sources of information provided for the task. A key factor to explain this relationship between debugging ability and switching frequency is the programmers' chunking ability (the ability to detect meaningful, hierarchical units in the code during problem solving). A high chunking ability is related to a robust mental representation of the program and therefore to a low need for duplicating references to the available representations.

The findings of this investigation support this view; however, they also suggest that this chunking ability effect might only be valid for programmers beyond a certain level of skill. Programmers with a low level of skill seem to concentrate too much on one representation only (the program code). Programmers at intermediate and high levels of skill switch more frequently between the available representations. However the relationship between switching frequency and programming ability is not linear; programmers of intermediate skill tend to switch more frequently between the available representations than those of high skill levels. It is possible that the chunking ability effect is only valid for novice programmers with intermediate and high skill levels (for details see Romero et al. (2003c)).

Individual differences. This investigation took into account three individual difference dimensions: the verbaliser-visualiser distinction (Oberlander, Stenning, & Cox, 1999), translation between representations ability (how good people are at translating between representational formalisms) (Oberlander et al., 1999), and *graphical literacy* (their level of familiarity with different representational formalisms) (Cox, 1999). From these dimensions, only graphical literacy was correlated to debugging accuracy. In general, people with a high level of graphical literacy were able to recognise which representations contained relevant information for the debugging problem at hand (for details see Cox, Romero, du Boulay, and Lutz (2003)).

These results suggest that it is the ability to decode the representations provided, more than modality preference or the capacity to translate information from one representation formalism into another, which is the key to successful programming performance when working in multi-representational environments.

Modality. This investigation found differences due to the modality of the available representations both in terms of programming performance and behaviour. Graphical representations seem to be more useful than textual ones when programmers with an intermediate level of experience face a challenging debugging task. Also, they seem to promote a more judicious representation use than textual ones in novice programmers with a high level of skill. These programmers switched more frequently in the

textual condition than in the graphical one. These results can be interpreted in several, possibly complementary, ways. One is that the specificity of graphical representations (Stenning & Oberlander, 1995) helps programmers to detect the meaningful units or chunks of the code, and in this way decreases the need for switching constantly between the available representations. Another interpretation is that as textual representations generally require more active search (Cox, 1999), this condition might have required greater working memory resources. This probably meant that participants were not able to hold as much information in working memory about the program as in the graphical condition. As a result, representation switching was more frequent when working with textual visualisations (for details see Romero et al. (2003b) and Romero et al. (2003c)).

Perspective. The findings of this investigation support the view that Java as an Object-Oriented language highlights data structure information whilst obscuring control-flow. Programmers found control-flow errors more difficult to find than those related to data structure (for details see Romero et al. (2002)). However, this finding should be taken with caution as this study made clear that software errors are multi-faceted and so classifying them as belonging to only one information type is usually not possible.

Principles for program comprehension aids

The empirical studies were largely carried out using novice computer programmers. Accordingly, we will place emphasis in our discussion of design principles for program comprehension aids on those suitable for novices.

During our work it became apparent that one cannot talk about novice computer programmers as a uniform group. Their behaviour varies according to how much of a novice they are. More experienced students interact more and make more intelligent use of the programming environment. It is therefore clear that providing facilities and having people use them are two different things. In particular, poorly performing novices tend to either ignore representations (other than the code), or get lost in the representations. It is hard to know quite how to interpret this finding, but it seems plausible that, at an early stage in their programming learning, students find the other representations too difficult to decode, so either ignore them, or spend very long lengths of time trying to understand them. Another finding was that experts are much more dependent than novices on the programming environment, and get frustrated by (novice, or toy) environments that do not provide all the facilities they are used to.

These findings suggest that programming environments need a range of representations, from very simple and intuitive for beginners, to the full complexity expected by more experienced programmers. Also, novices could benefit from contextual help that explained the representation formalisms employed in these environments.

The finding that graphical representations might be more useful than textual ones when programmers with an intermediate level of experience face a challenging debugging task suggests that both graphical and textual representations are needed. Programmers should be able to select the one they consider as more appropriate for the problem at hand. The general principle here is that the representations employed in programming environments should be of a flexible nature enabling them to be tailored both to the individual's modality preference (at that point in their programming career), and enabling various stages of simplification to be applied, making them suitable for both novice and expert use.

Our work has implications not just for the design of (novice) programming environments, but also for the way programming is taught. One way to interpret some of our findings (e.g. that novices seem to have difficulty understanding alternative representations) is that explicit instruction on this needs to be given as part of their programming course. Furthermore, since novices seem to have trouble working with dynamic debugging environments (Romero et al., 2003b), perhaps explicit teaching of debugging strategies within these environments also needs to be incorporated into the curriculum.

We developed some initial ideas as to how an intelligent learning environment for debugging might exploit the information logged by the SDE to advise on debugging strategy (du Boulay et al., 2003).

Finally, a side effect of the restricted focus technology seemed to be that it allowed people to leave *visual bookmarks* on the several representations they were working with. Each SDE window ‘remembered’ the position of its most recently visited unblurred spot, and this seemed to be of help to programmers, especially when they were switching quickly between the different representations in the environment (for details see Romero et al. (2002)). This notion of a visual bookmark should be easy to implement in any multi-representational environment as a feature that can be turned on or off and can be of help, for example, when comparing information on different windows of the environment.

Research Impact and Benefits to Society

The work has been presented at a wide range of international conferences and received favourable comment. Follow-on grant applications are in preparation. The work has also influenced programming teaching practice at the University of Sussex.

Dr P. Romero was enabled to progress from his named researcher position on this grant to a lectureship in the new department of Informatics at Sussex.

Further Research or Dissemination Activities

The work was presented by Dr Pablo Romero at the 2002 IEEE Symposia on Human Centric Computing languages and Environments (in Arlington, Virginia, USA), at the 2002 Diagrammatic Representation and Inference conference (in Callaway Gardens, Georgia, USA), at the 15th Workshop of the Psychology of Programming Interest group (in Keele), and at seminars at the School of Informatics, University of Northumberland at Newcastle and at The School of Cognitive and Computing Sciences, Sussex University. The work was presented by Professor Benedict du Boulay at a workshop on Innovations in Teaching programming at AIED2003 (in Sydney) and at a symposium on multiples representations at EARLI2003 (in Padua, Italy). In addition the work was presented at a seminar by Dr Richard Cox at the ESRC centre for Research in development, Instruction and Training (CREDIT), University of Nottingham.

The work has been accepted for presentation at the 2003 IEEE Symposia on Human Centric Computing languages and Environments (in Auckland, New Zealand). A paper on the empirical work has been submitted to the International Journal of Human-Computer Studies, and further papers based on analysis of the results from the second experiment are in preparation.

Two follow on EPSRC grant applications are in preparation. One will concentrate on investigating the possible advantages of programmers reasoning with self-constructed representations; the other will focus on taking a closer look at the process of representation interpretation in programming, characterising its subtasks and identifying problems encountered by novices and common errors they make.

References

- Ainsworth, S., Wood, D., & Bibby, P. (1996). Co-ordinating multiple representations in computer based learning environments. In Brna, P., Paiva, A., & Self, J. (Eds.), *Proceedings of the 1996 European Conference on Artificial Intelligence on Education*, pp. 336–342 Lisbon, Portugal.
- Blackwell, A., Jansen, A., & Marriott, K. (2000). Restricted focus viewer: a tool for tracking visual attention. In Anderson, M., Cheng, P., & Haarslev, V. (Eds.), *Theory and Application of Diagrams. Lecture Notes in Artificial Intelligence 1889*, pp. 162–177. Springer-Verlag.

- Corritore, C. L., & Wiedenbeck, S. (1999). Mental representations of expert procedural and object-oriented programmers in a software maintenance task. *International Journal of Human Computer Studies*, 50, 61–83.
- Cox, R. (1999). Representation construction, externalised cognition and individual differences. *Learning and Instruction*, 9, 343–363.
- Cox, R., Romero, P., du Boulay, B., & Lutz, R. (2003). Differential effects of representational knowledge on program comprehension and debugging. In *Submitted to Diagrammatic Representation and Inference. Third International Conference, Diagrams 2003*.
- Crosby, M., & Stelovsky, J. (1989). Subject differences in the reading of computer algorithms. In Salvendy, G., & Smith, M. J. (Eds.), *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, pp. 137–144. Elsevier science publishers B.V., Amsterdam, The Netherlands.
- 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 van Someren, M. W., Reimann, P., Boshuizen, H. P. A., & de Jon, T. (Eds.), *Learning with Multiple Representations*, pp. 9–40. Elsevier Science, Oxford, U.K.
- du Boulay, B., Romero, P., Cox, R., & Lutz, R. (2003). Towards a debugging tutor for object-oriented environments. In Alevy, 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.
- Gilmore, D. J., & Green, T. R. G. (1984). Comprehension and recall of miniature programs. *International Journal of Man-Machine Studies*, 21(1), 31–48.
- Green, T. R. G. (1989). Cognitive dimensions of notations. In Sutcliffe, A., & Macaulay, L. (Eds.), *People and Computers V*, pp. 443–460. Cambridge University Press, Cambridge.
- Oberlander, J., Stenning, K., & Cox, R. (1999). Hyperproof: Abstraction, visual preference and modality. In Moss, L. S., Ginzburg, J., & de Rijke, M. (Eds.), *Logic, Language, and Computation, Vol. II*, pp. 222–236. CSLI Publications.
- Pennington, N. (1987). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19, 295–341.
- 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., Cox, R., du Boulay, B., & Lutz, R. (2003a). A survey of representations employed in object-oriented programming environments. *Journal of Visual Languages and Computing*, 14(5).
- Romero, P., du Boulay, B., Cox, R., & Lutz, R. (2003b). Java debugging strategies in multi-representational environments. In Petre, M. (Ed.), *Psychology of Programming Interest Group 15th Workshop*, pp. 421–434.
- Romero, P., du Boulay, B., Lutz, R., & Cox, R. (2003c). The effects of graphical and textual visualisations in multi-representational debugging environments. In Burnett, M., & Grundy, J. (Eds.), *2003 IEEE Symposia on Human Centric Computing Languages and Environments*. IEEE press, Auckland, New Zealand.
- Romero, P., Lutz, R., Cox, R., & du Boulay, B. (2002). Co-ordination of multiple external representations during java program debugging. In Wiedenbeck, S., & Petre, M. (Eds.), *2002 IEEE Symposia on Human Centric Computing Languages and Environments*, pp. 207–214. IEEE press, Arlington, Virginia, USA.

- Stenning, K., & Oberlander, J. (1995). A cognitive theory of graphical and linguistic reasoning: logic and implementation. *Cognitive Science*, *19*(1), 97–140.
- Vessey, I. (1985). Expertise in debugging computer programs: a process analysis. *International Journal of Man-Machine Studies*, *23*, 459–494.
- Widowsky, D., & Eyerth, K. (1986). Comprehending and recalling computer programs of different structural and semantic complexity by experts and novices. In Willumeit, H. P. (Ed.), *Human Decision Making and Manual Control*. Elsevier, Amsterdam, Holland.
- Wiedenbeck, S., & Ramalingam, V. (1999). Novice comprehension of small programs written in the procedural and object-oriented styles. *International Journal of Human Computer Studies*, *51*, 71–87.
- Wiedenbeck, S., Ramalingam, V., Sarasamma, S., & Corritore, C. L. (1999). A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, *11*, 255–282.