

CRUSADE - Coordination of multiple external Representations in program UnderStAnding and DEbugging: Research proposal

1 Summary

The use of multimedia systems and graphical interfaces has made the use of multiple external representations commonplace for computerised learning environments. In the field of computing, novice and professional software development environments often offer multiple views or abstractions of the program code, and so, multiple views to aid programming tasks is an issue for programmers of all levels. However, there is little theoretical knowledge about the way these multiple representations influence the comprehension of computer programs.

In particular, we are interested in how different factors affect the way users co-ordinate representations when performing programming tasks. Some of the factors that will be considered are: the different types of information or perspective afforded by the representations, information modality and individual differences such as cognitive style.

This research will study novice programmers of Java when using software visualisation representations for program comprehension tasks. We are interested in how the co-ordination of these external representations influences 1) the form of their mental representations and 2) their comprehension strategies.

The results of this project will provide an empirical basis for assessing the benefits of multiple representations for teaching programming and will inform the design of effective programming environments.

Keywords: Diagrammatic representations, Object-Oriented Programming, Psychology of programming, Cognitive science.

2 Introduction

When trying to perform a programming activity in everyday settings, programmers normally work with other external representations as well as the program code. Some of these external representations occur in debugging packages, prototyping and visualisation tools in software development environments, or are included as part of internal and external documentation. Therefore, programming normally requires the co-ordination of multiple representations.

Probably the most typical case, at least for beginner programmers, of co-ordination of external representations in programming is working with debugging packages, a common example of a visualisation tool. Novice programmers often spend a good amount of their learning time attempting to understand the behaviour of programs when trying to discover errors in the code. To perform this task, novices normally work with both the program code and the debugger output, trying to co-ordinate and make sense of these representations. Yet studies of program comprehension have not, to the best of our knowledge, addressed the issue of using multiple external representations for the program comprehension task.

Three important aspects to consider regarding the co-ordination of multiple representations are the issue of sentential versus graphical representations, the different information types implicit in programs and the cognitive style of programmers. The first aspect refers to co-ordinating representations which are basically propositional with those that are mainly diagrammatic. It is not clear whether co-ordinating representations of 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.

The second aspect refers to co-ordinating representations that highlight either the same or different programming information types. Computer programs are information structures that comprise different types of information, and programming notations usually highlight some of these aspects at the cost of obscuring others. It is an open issue whether co-ordinating notations that highlight different information types will be more beneficial to programmers than working with those that highlight the same ones.

The third aspect has to do with individual differences of programmers. Cognitive preference is highly relevant to the issue of co-ordination of external representations. The question here is which characteristics make some people more successful at co-ordinating representations than others.

This project proposes to explore these issues. In particular, it focuses on investigating which conditions promote an appropriate co-ordination of multiple representations when comprehending a program and the role that this co-ordination has in the development of program comprehension skills.

2.1 Co-ordinating multiple external representations

The use of multiple external representations in learning environments offers several advantages over learning within a single representation. Two important advantages are that specific information can be best conveyed in a specific representation and that as expertise is normally considered as possessing and co-ordinating multiple representations of the domain, the learning process should encourage the development of multifaceted representations of this kind (de Jon, Ainsworth, Dobson, van der Hulst, Levonen, & Reimann, 1998).

The use of multiple external representations in the training stage might promote these multifaceted representations. There are different dimensions according to which multiple external representations can be co-ordinated. Two relevant dimensions are modality and perspective (de Jon et al., 1998).

Modality refers to the particular form of expression that is used to present information. A typical distinction here is between propositional and

diagrammatic representations, although these two terms can be thought of as situated at the extremes of a continuum that comprises representations with different degrees of ‘graphicality’ (Cheng, Lowe, & Scaife, 1999). Program code, for example, cannot be regarded as fully propositional, because it uses location conventions to enhance its comprehension (there is normally a line-per-instruction format and it makes extensive use of indentation). In the taxonomy of graphic languages developed by Twyman (1979), program code would not be an example of pure linear text but a hybrid category of text between a list and a linear branching configuration.

Perspective refers to the different ‘views’ that can be adopted to look at a domain. The ability to build several perspectives of a domain and to switch between them during problem solving as a means to cope with complexity is directly related to domain expertise (Nix & Spiro, 1990).

However, one potential problem of working with multiple external representations is the difficulty of co-ordinating them. Here, the empirical studies offer mixed results which suggest that the effects of multiple representations are related to user characteristics (Oberlander, Stenning, & Cox, 1999) and also that testing these effects might not be so simple, at least in science teaching (Swaak & de Jong, 1996).

2.1.1 Information modality

Representations of differing modality are a common case of multiple external representations that support complementary processes (Ainsworth, 1999). For example, diagrams, unlike propositional representations, exploit perceptual processes by grouping relevant information together and therefore make the search and recognition of information easier. Another difference between propositional and diagrammatic representations is that the former permit the expression of abstraction or indeterminacy while the latter compel the representation of specific information (Stenning & Oberlander, 1995). This specificity of graphical representations allows some inferences to be more tractable.

As mentioned above, there is not a clear-cut division between diagrammatic and propositional representations. Instead, there are gradations of ‘graphicality’. In general, the more different a degree of graphicality external representations exhibit, the more difficult it is for students to co-ordinate them (Ainsworth, Wood, & Bibby, 1996). One important factor when considering the benefits of representations of different modalities is individual differences. Oberlander et al. (1999) claim that individual preference for diagrammatic representations is related to the ability to translate between modalities. This seems to be in agreement with Ainsworth, Wood, and O’Malley’s (1998) conclusion that the design of environments that require the co-ordination of different modalities has to pay special attention to providing cues to support the process of translation between representations. Hope for the idea of teaching modality translation skills comes from Cox (1999). He claims that this ability is probably not an immutable cognitive trait and is therefore responsive to educational intervention.

Although programmers normally have to coordinate representations of different modalities, there has not been much research on these issues in the area of programming. One of the few examples here is the GIL

system (Merrill, Reiser, Beekelaar, & Hamid, 1992), which attempts to provide reasoning-congruent visual representations in the form of control-flow diagrams to aid the generation and comprehension of LISP, a textual programming language. Merrill et al. claim that this system is successful in teaching novices to program in this language; however, their work did not compare co-ordination of the same and different modalities.

Other studies in the area have been concerned with issues related to the format of the output of debugging packages (Patel, du Boulay, & Taylor, 1997; Mulholland, 1997). Those studies have offered conflicting results about the co-ordination of representations of different modalities. Patel et al. (1997) found similar performance for subjects working with representations of the same and different modalities while Mulholland (1997) reported poor performance for those working with different modalities. In both cases, participants worked with the program code and with the debugger's output. The debugger notations used by both of these studies were mostly textual. The only predominantly graphical debugging tool used by these studies was TPM (Eisenstadt, Brayshaw, & Paine, 1991). While the performance of Patel et al.'s participants was similar for the textual debuggers and TPM, the subjects of Mulholland's study found working with TPM more difficult. One important difference between these two studies is that while the former used static representations, the later employed a visualisation package and therefore dynamic representations. However, other factors might be playing a role in these conflicting results. First, it is not clear that the graphical debugger (TPM) displayed a similar amount of information to its textual counterparts. Other important factors that Patel et al. (1997) identified are the choice of experimental materials (the programs employed in the experiments) and the degree to which the information required for the experimental task is hidden or implicit in the alternative representation. An important issue is the degree to which the above work on LISP and Prolog generalises to an Object-Oriented language like Java.

2.1.2 Perspective

Representations with differing perspectives are used when a single representation would be insufficient to carry all the information about the domain, or when attempting to combine all relevant information into a single representation would over-complicate the learner's task (Ainsworth, 1999). As with modality, co-ordinating representations of different perspective seems to be more complicated than co-ordinating those of the same perspective (Gruber, Graf, Mandl, Renkl, & Stark, 1995).

In programming it has been established that programs can be looked at from different perspectives (Pennington, 1987b); and experienced programmers, when comprehending code, are able to develop a mental representation that comprises these different perspectives or *information types*, as well as rich mappings between them (Pennington, 1987a; Ormerod & Ball, 1993). Some of these different information types are: function, structure, operations, data-flow and control-flow.

Gilmore and Green (1984) proposed the match-mismatch hypothesis, suggesting that every notation highlights some information at the cost of obscuring some other. In procedural languages, for example, it is relatively

easy to detect control-flow information, and relatively difficult to access function relations (Pennington, 1987b; Corritore & Wiedenbeck, 1991). For different programming languages the information types highlighted and obscured are different. For Prolog, a declarative programming language, it is data structure information which is accessible (Bergantz & Hassell, 1991; Romero, 1999), while according to Wiedenbeck and Ramalingam (1999), novices of C++ working with small programs tend to develop a mental representation strong in function-related knowledge, but have problems detecting operations and control-flow information.

To the best of our knowledge, there are no studies that have looked at perspective co-ordination in programming. Mulholland (1997), for example, employed a language that highlights data structure information (Prolog) and visualisation tools that highlight data-flow and control-flow information; however, he was more interested in developing an evaluation methodology than in perspective co-ordination.

2.2 Java and Object-Oriented programming

This project will investigate the co-ordination of modality and perspective for student programmers performing comprehension-related tasks in Java, an Object-Oriented (OO) programming language. Java, and the OO paradigm in general, have undergone an explosive growth in both professional software development and novice programmer training. However, there is only a small number of empirical studies of the OO paradigm, and most of these are concerned with software design issues (Détienne, 1997). Also, the majority of methodologies and tools for Java have been developed for professional programmers working at the software design stage (Reed, 1998).

Novices of OO languages have a hard task. Besides having to learn OO concepts, they also have to master procedural aspects of programming. Although OO languages seem to highlight functional information (Wiedenbeck & Ramalingam, 1999; Corritore & Wiedenbeck, 1999), it is not clear whether novice programmers working with medium size programs find this an easy task (Wiedenbeck, Ramalingam, Sarasamma, & Corritore, 1999), specially because as program size increases, functional information tends to become diffuse. In our experience, students of Java have problems conceptualising the concept of an object, understanding, for example, that instances of objects have to be created and initialised and that objects have a state as well as a defined behaviour. They also have problems following control flow through multiple objects and understanding the difference between the different kinds of variables in Java. To make things even harder, there are not many tools to aid the comprehension of programs in Java. For instance, although there are visualisation packages (like VisiComp or CodeVizor) and debugging tools included in programming environments (like Borland JBuilder and CodeWarrior) these are aimed at professional programmers. Also, there does not seem to be a standard for the kinds of representations used by visualisation environments designed as an aid to comprehension related tasks. The Unified Modelling Language (UML) (Booch, Rumbaugh, & Jacobson, 1998), for example, proposes several types of views of the program, but these form part of a methodology for software design. There are also other types of representations used by algorithm animations packages for Java (Lahtinen,

Sutinen, & Tarhio, 1998), by the visualisation and programming environments mentioned above or by teaching methodologies and tools for other OO languages like Smalltalk (Chee, 1995) or Eiffel (Rist, 1996). It is an open issue whether these kinds of representations can be applied successfully for Java program comprehension.

From this brief review of the area, it can be seen that there is a need for more research about the kinds of external representations that programmers encounter and the factors that influence their co-ordination.

3 Programme and methodology

3.1 Overall aims of the project

- to investigate the role that perspective, modality and individual differences such as cognitive style play in the co-ordination of multiple external representations in novice program comprehension.
- to develop a set of design principles for program comprehension aids, and particularly for software visualisation packages for Java.
- to develop a computerised experimental tool that will include a prototype visualisation environment for Java suitable for learners.

3.2 Methodology

The core work of this project will take the form of experiments involving novice Java programmers working with a visualisation-like experimental tool performing debugging tasks. This tool will provide code views of different kinds. These different kinds of code view will include propositional and diagrammatic; static and dynamic; and views from different perspectives or information types. This will allow us to compare the co-ordination of representations for dimensions like modality and perspective within the context of a common programming activity like debugging.

3.3 Sufficient timeliness and novelty

The work proposed is timely in that, as mentioned above, there do not seem to be standards for the kinds of representations used by visualisation environments in the Object-Oriented paradigm, and more specifically for the Java language. Although there is a long tradition of programmers working with multiple external representations when performing common programming tasks, we are not aware of studies about this issue in the program comprehension area. Finally, empirical studies of programmers have not, to the best of our knowledge, looked at the issue of cognitive preference (individual differences). This aspect seems to be of particular relevance for the instruction process and therefore for novice programmers.

3.4 Programme of work

The experiments will explore how the co-ordination of external representations influences structural and strategic aspects of programming knowledge. In

particular, it will compare the mental representations and strategies employed by programmers co-ordinating different and similar external representations. It will investigate the dimensions of modality and perspective and will analyse the role that individual differences play in these types of co-ordination. This research will focus on beginner programmers of Java working with faulty program code (the *code representation*), a sample output of a correct program version (the *correct output representation*) and a representation of the buggy code similar to the ones produced by visualisation tools (the *visualisation representation*). This study will comprise two experiments. Although in both cases the experimental session will be administered automatically by an experimental program, in the first one participants will work with static representations (as in Patel et al.'s (1997) experiment) and in the second with dynamic ones (as in Mulholland's (1997) experiment). In both cases the user/system interactions will be dynamically logged. The main reason for performing this investigation through two experiments has to do with the conflicting results reported in the Patel et al. (1997) and Mulholland (1997) studies. Factors which account for this difference of results probably include the difficulty of using the dynamic graphical representations of the TPM debugger employed in each of these studies. Performing experiments with static and dynamic representations, but with otherwise similar settings, will offer clues about the difficulty of co-ordinating representations of different modality. Another reason to perform this study in two parts is that the results of the first experiment can inform methodological aspects and design issues for the experimental environment that will manage the dynamic representations in the second experiment. This first experiment will allow us, for example, to investigate the suitability of different formats for the visualisation representation with relatively small effort.

3.4.1 The experiments

These two empirical studies will have similar settings. Both experiments will require the co-ordination of representations according to modality and perspective. The visualisation and code representations will be similar or different according to the referred dimensions. For the case of modality, as both the code and correct output representations are propositional, the visualisation representation will be mainly propositional or mainly diagrammatic. For the case of perspective, because Java as an OO language tends to highlight function and to obscure control-flow information (Wiedenbeck & Ramalingam, 1999), the visualisation representation will be functional or control-flow related. It will also be interesting to compare small and medium size programs, because novices find it difficult to detect function as the size of the program increases (Wiedenbeck et al., 1999). It is not clear yet which conventions will be used to represent these different information types in the visualisation representations; formats considered will include, among others, those proposed by the UML language (Booch et al., 1998), by algorithm animation packages for Java (Lahtinen et al., 1998) and by teaching tools and methodologies for other OO languages (Chee, 1995; Rist, 1996). In both experiments participants will be classified according to a *verbal/visual* preference and they will also perform a test to determine the degree to which

they are able to translate between representations (Lohse, Walker, Biolsi, & Rueter, 1991; Cox, 1996). These two classifications will be similar to those employed by Oberlander et al. (1999). They showed that co-ordinating between representations of different modality successfully had more to do with the ability to translate between representations (the *transmodal hypothesis*), than with the simple verbaliser-visualiser distinction (the *visual preference hypothesis*). The experiments proposed here will compare the co-ordination of similar and different modalities and will therefore be useful to test these hypotheses further. According to the visual preference hypothesis, the preferred co-ordination mode for verbalisers will be between propositional representations (the one involving a textual visualisation), while for visualisers it will be the one that includes a diagrammatic representation (the one with a graphical visualisation). On the other hand, according to the transmodal hypothesis, participants categorised as ‘good translators’ would be more comfortable with both kinds of co-ordination since in both cases they have to translate between different representations.

In each experiment there will be two experimental tasks. The first task will involve finding errors in program code. Programs will contain errors related to the different information types. In the first task the experimental environment will allow participants to look at only one external representation at a time in order to track focus of attention. Also, subjects will be asked to verbalise their thoughts so that a verbal protocol analysis can be performed. The patterns of inspection of the external representations will be compared to find out in what cases participants performed switches between different representations. Also, the participants’ utterances will be classified according to the information type they refer to, similarly to Mulholland’s (1997) experiment (function, control-flow, structure, operations, data-flow or mixed). Both the patterns of inspection and verbal protocols will be related to debugging performance. In general, it is expected that good translators will develop a more consistent switching strategy, will exhibit a high frequency of mixed type utterances and will achieve higher debugging performance, specially for errors of the information types highlighted by either the code or the visualisation representation. A high frequency of utterances of the mixed type would indicate that this group was trying to build rich mappings between the different information types.

In the second task, participants will answer several yes-no comprehension questions referring to the different information types comprised in the debugged program. This way of measuring comprehension performance has been used in several program comprehension studies (Pennington, 1987b; Corritore & Wiedenbeck, 1991; Wiedenbeck & Ramalingam, 1999; Corritore & Wiedenbeck, 1999) and is similar to the one suggested by Swaak and de Jong (1996). The accuracy and response time to these questions will be recorded automatically. It is expected that participants who were successful in the debugging task will do well in general in the question answering task. Also, their scores for questions related to the information types highlighted by either the code or the visualisation representations should be higher than those related to other information types. As Java programs usually highlight functional information, if they worked with a function related visualisation representation, it will be only this information type which will stand out as the one with the highest scores. On the other hand, if they worked with a

control-flow visualisation representation, both function and control-flow questions are expected to have high scores. In between these two subtasks, subjects will be shown a correct version of the code so that they will not carry an incorrect model of the program to the question answering subtask.

3.4.2 The experimental tool

The second experiment will employ a prototype dynamic visualisation tool. This tool will provide a simple step-and-trace account of the program behaviour according to the different dimensions considered (modality and perspective). Although different representations will be employed by the different dimensions considered, the way of handling the visualisation tool will be similar for these different representations. This experimental tool will also record user/system interactions dynamically.

3.5 Relevance to beneficiaries

The results of these two experiments will inform the design of learning environments for programming (and programming tools such as visualisation packages) if they want to take advantage of multiple representations for effective teaching. Specifically, they will provide an empirical basis for assessing the benefits of multiple representations for teaching programming and will inform the design of effective programming environments. The findings of this project will feed into knowledge about the design of learning environments for programming, and about the design of programming tools in general. Visualisation software designers and developers of programming tools in general are likely to benefit. This research will also be of interest to the Object-Oriented, psychology of programming and diagrammatic representations communities.

References

- Ainsworth, S. (1999). The functions of multiple representations. *Computers & Education*, 33(2-3), 131–152.
- 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.
- Ainsworth, S., Wood, D., & O'Malley, C. (1998). There is more than one way to solve a problem: evaluating a learning environment that supports the development of children's multiplication skills. *Learning and Instruction*, 8(2), 141–157.
- Bergantz, D., & Hassell, J. (1991). Information relationships in PROLOG programs: how do programmers comprehend functionality?. *International Journal of Man-Machine Studies*, 35, 313–328.

- Booch, G., Rumbaugh, J., & Jacobson, I. (1998). *The Unified Modeling Language User Guide*. Addison-Wesley.
- Chee, Y. S. (1995). Cognitive apprenticeship and its application to the teaching of smalltalk in a multimedia interactive learning-environment. *Instructional Science*, *23*(1-3), 133-161.
- Cheng, P., Lowe, R., & Scaife, M. (1999). Cognitive science approaches to understanding diagrammatic representations. *AI Review*, *In press*.
- Corritore, C. L., & Wiedenbeck, S. (1991). What do novices learn during program comprehension?. *International Journal of Human-Computer Interaction*, *3*(2), 199-222.
- 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. (1996). *Analytical reasoning with multiple external representations*. Ph.D. thesis, University of Edinburgh, Edinburgh, Scotland, U.K.
- Cox, R. (1999). Representation construction, externalised cognition and individual differences. *Learning and Instruction*, *9*, 343-363.
- 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.
- 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.
- Eisenstadt, M., Brayshaw, M., & Paine, J. (1991). *The Transparent Prolog Machine*. Intellect, Oxford, England.
- Gilmore, D. J., & Green, T. R. G. (1984). Comprehension and recall of miniature programs. *International Journal of Man-Machine Studies*, *21*(1), 31-48.
- Gruber, H., Graf, M., Mandl, M., Renkl, H., & Stark, R. (1995). Fostering applicable knowledge by multiple perspectives and guided problem solving. In *Conference of the European Association for Research on Learning and Instruction* Nijmegen, The Netherlands.
- Lahtinen, S. P., Sutinen, E., & Tarhio, J. (1998). Automated animation of algorithms with eliot. *Journal of Visual Languages and Computing*, *9*, 337-349.
- Lohse, G., Walker, N., Biolsi, K., & Rueter, H. (1991). Classifying graphical information. *Behaviour and Information Technology*, *10*(5), 419-436.

- Merrill, D. C., Reiser, B. J., Beekelaar, R., & Hamid, A. (1992). Making processes visible: scaffolding learning with reasoning-congruent representations. *Lecture Notes in Computer Science*, 608, 103–110.
- 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.
- Nix, D., & Spiro, R. J. (1990). Cognitive flexibility and hypertext: Theory and technology for the non-linear and multi-dimensional traversal of complex subject matter. In Spiro, R. J., & Jehng, J.-C. (Eds.), *Cognition, Education and Multi-media: Exploring Ideas in High Technology*, pp. 163–205. Erlbaum, Hillsdale, New Jersey.
- 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.
- Ormerod, T. C., & Ball, L. J. (1993). Does design strategy or programming knowledge determine shift of focus in expert Prolog programming?. In *Empirical Studies of Programmers, fifth workshop*, pp. 162–186 Norwood, New Jersey. Ablex.
- 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. (1987a). 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. (1987b). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19, 295–341.
- Reed, P. (1998). The unified modeling language takes shape. *DBMS*, 11(7), 47–58.
- Rist, R. S. (1996). Teaching eiffel as a first language. *Journal of Object-Oriented programming*, 9, 30–41.
- Romero, P. (1999). Focal structures in prolog. In Green, T., Abdullah, R., & Brna, P. (Eds.), *Psychology of Programming Interest Group 11th Workshop*, pp. 7–17.
- Stenning, K., & Oberlander, J. (1995). A cognitive theory of graphical and linguistic reasoning: logic and implementation. *Cognitive Science*, 19(1), 97–140.
- Swaak, J., & de Jong, T. (1996). Measuring intuitive knowledge in science: The development of the what-if test. *Studies of Education Evaluation*, 22, 341–362.

- Twyman, M. (1979). A schema for the study of graphic language. In Kolars, P. A., Wrolstad, M. E., & Bouma, H. (Eds.), *Processing of Visible Language*, pp. 117–150. Plenum Press, New York.
- 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.