Mental Representation and Imagery in Program Comprehension

Raquel Navarro-Prieto

Experimental Psychology Department

Campus de la Cartuja
18071 Granada, Spain
rnavarro@goliat.ugr.es
http://www.ugr.es/~rnavarro

Jos— J. Ca™as

Experimental Psychology Department
Campus de la Cartuja
18071 Granada, Spain
delagado@goliat.ugr.es
http://www.ugr.es/~delagado

ABSTRACT

This paper studies the role of imagery in program comprehension. With this goal we investigated whether theories of mental models from Psychology of Programming (e.g., Pennington's Two Stages Theory) could be expanded to account for the effect of imagery. Given the basic research in image processing, our hypothesis is that imagery would allow a quicker access to the functional (Data Flow) information of programs. Then, Visual Programming Languages should allow for quicker construction of a mental representation based on Data Flow relationships of a program than procedural languages. To test this hypothesis we ran an experiment where we accessed the mental model of C and spreadsheet programmers in different program comprehension situations. The results showed evidence that the spreadsheet programmers developed Data Flow based mental representations in all situations while C programmers seemed to access to a Control Flow based mental representation first.

Keywords

Imagery, mental model, Visual Programming Language, Spreadsheets, C, program comprehension.

INTRODUCTION

Program comprehension is a complex cognitive skill which involves the acquisition of a mental representation of program structure and function. Imagery is a cognitive process that can play an important role in how this mental representation is acquired. Our research is aimed at studying this role of imagery in program comprehension by bringing together the data and methodology from Psychology of Programming and image processing research.

Basic research in image processing indicated that pictorial material would give faster access to semantic information than verbal material (Bajo, 1988). Also research on the effect of visual aids in text comprehension (Mayer and Gallini, 1990) and HCI learning (Navarro, Cañas and Bajo, 1996) has shown that visual aids enhance learning only when they could facilitate access to the meaningful information for a given situation. All together, these data support the hypothesis that visual aids could enhance computer use and the acquisition of mental representations because they would improve access to meaningful information.

In the last decade the mental model approach to text understanding (van Dijk & Kintsch, 1983) has been successfully applied in Psychology of Programming to explain the cognitive stages underlying program comprehension (Pennington, 1987; Corritore & Wiedenbeck, 1991; Burkhardt, Détienne, Wiedenbeck, 1997). According to Pennington's Two Stage Theory (1987) programmers go through two phases when they try to understand a program. In the first stage, they develop a knowledge structure representation (program model) based on the Control Flow relationships (i.e control patterns like loops or conditional patterns). In the second stage, under appropriate task conditions, programmers develop a plan knowledge representation (domain model) based on the Data Flow. This representation would contain the main program functions and the key information to understand what the program does. It also includes information about the programming situation.

Several variables from the situation influence how and when programmers go through these stages. Two of these variables are notations' visual characteristics (Scalan ,1989) and programming language (Gilmore and Green, 1988) Visual languages (VPLs) seem to facilitate program comprehension as compared to textual languages. This effect points to the role that imagery could play in program comprehension. However, not much research has been done in this issue that would allow us to expand text comprehension theories to explain how VPL programmers develop their mental representations.

Our hypothesis is that IF the role of imagery is to enhance the access to the meaningful information THEN VPLs should allow quicker access to the Data Flow information of a program than procedural languages. Therefore, visual programmers should more quickly develop a representation based on Data Flow relationships, even when performing a simple comprehension task, in comparison with other non visual programming languages.

Recent research showed that the type of comprehension task that programmers perform influences the mental representation that they acquired. For example, Détienne (1996) has shown

that read-to-do (e.g. for modification) and read-to-recall (e.g. for documentation) tasks affect differently the mental model construction. Pennington's theory could explain this effect of type of comprehension task. For performing a relatively easy task, like reading a program, programmers only go through the first stage. However, to modify the program they would go through the second stage, understanding the Data Flow structure.

We hypothesised that textual language programmers would access Data Flow information only when they perform a difficult task. However, visual language programmers would access this information even when they perform an easy task. To test this hypothesis we designed an experiment in which C and Spreadsheet programmers are assessed on their mental representations of programs, under different comprehension conditions. We are interested in studying how the visual format of spreadsheets affects mental model construction, both in a simple reading and a modification task.

EXPERIMENT

Procedure

First, programmers were asked to fill in a questionnaire about their programming experience. Then, programmers of C or Spreadsheets were asked to read two programs (the 'Easy Task') and modify two other programs (the 'Difficult Task'). They were instructed to read the program until they thought they could understand the program (Easy Task) or could make the required modification (Difficult task). The time limit for both types of comprehension tasks was 10 minutes After reading or modifying a program, they performed two tasks designed to elicit the mental representation of the program's structure that they have acquired:

Primed Recognition task: This task has been widely used in image processing research. The motivation for using this task is that it has been shown to be effective in testing whether the subject's mental representations are based on a hypothesised relationship. In each trial of this task, subjects were presented with a program segment (target) taken either from the program that they had read or modified, or from a different program. Their task was to decide as quickly as possible whether or not the segment was part of the program they had already seen. The target segment was preceded by another program segment (prime). The underlying assumption is that knowledge is organised in networks where the activation from one node spreads to nearby nodes. Therefore if the prime and target are related in the mental network, the activation of the prime would facilitate the activation of the target. The critical manipulation is the prime-target relationship. To test if the mental model developed by the subjects were based on data or Control Flow relationships, two theoretical networks were constructed for each program (one Control Flow network and another Data Flow network). Then, there were four priming conditions:

1. Data Flow related Condition: A target segment in the test is preceded by a prime close in the theoretical Data Flow network, and far in the Control Flow theoretical network.
2. Control Flow related Condition: A target segment in the test is preceded by a prime close in the theoretical Control Flow network, and far in the Data Flow theoretical network.
3. Unrelated condition: the target segment is preceded by a segment from the same program, but hypothesised to be far away in both the control and Data Flow theoretical networks.
4. Non-Program condition: the target segment was from a different program than the prime segment.

Half of the targets were part of the original program. The other half were not from the program, therefore the subject should reject them as not belonging to the original program. These non-program targets were constructed by modifying some program segments, so that the information in the modified fragment did not correspond with the information shown in the original program. In order to be sure that the Data Flow related condition is not affected by Control Flow information, and the other way around, two controls were done with the material. First, in the Data Flow conditions, we replaced the operations in these segments with dots. Following the same reasoning in the Control Flow conditions, information about the name of the variables were replaced with dots. Second, a distracter code fragment was presented with every target. The target and the distracter were identical except in an operation, in the case of the Control Flow condition, or a variable name in case of the Data Flow condition or one of these options for the rest of conditions. Therefore, the recognition decision had to be based on the exact recognition of the Data or Control Flow information from the program.

Primes were presented for 10 seconds, because we wanted to be sure that the subject had time to read them. During this time, no answer was required for the subject, who was instructed to carefully read the prime fragment. This time was calculated in a pilot study as the maximum time needed to read our fragments. After the prime disappeared, the next screen presented the target and distracter fragments until the subject responded. In this screen, subjects were asked to click on the fragment/s that they thought were from the original program, or if neither of them was from the original program, click the OK bottom to go to the next trial.

Recognition accuracy and time were recorded. We predicted a priming effect. Response times to the target segment preceded by a prime close in the network structure should be faster than response time (and with better accuracy) to the same target preceded by a prime which was not as close in the cognitive structure. This priming effect would be observed in the control and/or Data Flow conditions depending on the knowledge acquired by the subject.

Sorting Task: This task has been used successfully to access a subjects' mental model in programming

(e.g. Robertson and Yu, 1990). The subjects were presented with a program that they had read or modified previously, and were asked to group together the lines of code or the cells that they though were related to each other. To make a group they just needed to click on the lines/cells that they wanted to select. They could do all the groups that they found important. They were given some practice trials to this task with names of fruits and mountains.

Sixty-four subjects with different experience levels participated in the experiment. Thirty-two subjects were C programmers and thirty-two were Spreadsheet programmers.

Results

The data from the experience questionnaire were quantified by two expert programmers. We wanted the experience data from each subject to be sure that our results were not interacting with this variable, which has been shown to be important in developing mental representations in programming (Corritore & Wiedenbeck, 1991).

Sorting task results

Raw sorting data were transformed into proximity data by calculating the number of times that two segments of the program were grouped together by one subject. Therefore, for each subject there were four proximity matrices, one for each program (two belonging to the Modify Condition and two belonging to the Read Condition).

Those matrices were submitted to a Pathfinder analysis. This analysis resulted in networks with links among the segments, which represented relationships in the subjects' mental representation of the programs. The theoretical Control Flow and Data Flow networks were our criteria for measuring a subject's level of comprehension of Control/Data Flow information in the program. The PathFinder analysis provided us with a measure of the similarities among networks, called C (which range goes from 0 to 1). We calculated the C between each subject's network for each program and our theoretic control (four Cs) and Data Flow networks (four Cs). We averaged the Cs between the two modified and the two read programs for each subject. In total four C measures were calculated for each subject. The statistical analyses were done on these C values.

A factorial design, 2 X 2 X 2 (Language X Control vs. Data Flow Network X Comprehension task) was used. Language was a between-subjects variable and Control vs. Data Flow Network, and Comprehension task were within-subject variables.

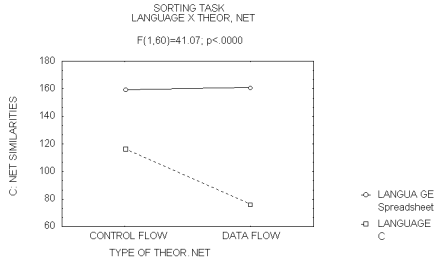

Figure 1. Sorting Results: Closeness (C parameter) of the C and Spreadsheet programmer networks to the Control and Data Flow theoretical networks.

Our results show significant effects of the programming Language ($F(1,60)= 108.934$, M.S.E= 0.0023, $p= 0.000$), the Prime Condition ($F(1,60)= 35.1022$, M.S.E.= 0.0006, $p= 0.000$) and the interaction between them ($F(1,60)= 41.75$, M.S.E.= 0.0006, $p= 0.000$, Fisher's Test of the Less Significant Difference (LSD) = 0.013). Spreadsheet programmer representation's were closer to both the control and Data Flow criteria (C = 0.16) than C programmers' mental representations (C = 0.09). Overall, control primes were closer to the theoretical networks, but that was due to the C programmers. As we can see in Figure 1, according to our hypothesis, C programmers have better networks for Control Flow information compared with Data Flow information. So, C programmers learned the Control Flow information better than Data flow information. On the other hand, Spreadsheet programmers seem to have developed good mental structures for both control and Data Flow information.

Primed recognition task

The number and average time of correct responses were recorded for positive and negative trials (where the target segment was part or not part of the studied program respectively) separately for each subject. A factorial design, 2 x 4 x 2 (Comprehension task x Prime Condition x Language) was used. Language was a between-subjects variable, and Comprehension Task and Prime Condition were within-subject variables. The level of experience of the subjects was again a covariant variable.

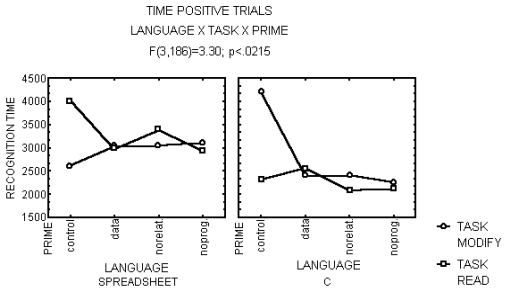

Figure 2. *Priming Recognition Results: Response times to Positive Trials, second order interaction: Programming Language by Comprehension Task and by Prime Condition.*

Positive Trials (in which the target segment was from the studied or modified program)

Recognition Time: The effect of Language was significant ($F(1,61)=7.332$; $p=0.008$). C programmers were faster (Average=2542 ms.) than Spreadsheet programmers (Average=3137 ms.). Close to significant was the interaction between the Language and Comprehension Task ($F(1,61)=3.49$, $p=0.067$), where C programmers showed larger differences between tasks than Spreadsheet programmers. However, this interaction was modulated by the effect of the significant second

order interaction of Prime Condition by Comprehension Task and Language ($F(3,186)=3.30$; $p<0.021$, LSD = 1265). When C programmers modified a program, Control primes slowed down recognition times compared with Data, Program Unrelated and Non Program. There were no differences among primes when C programmers had to read a program. For spreadsheet programmers Control primes tended to slow down recognition after reading, and made recognition faster after modifying a program faster.

Accuracy: There was no significant effects of any of the manipulated variables.

Negative Trials (in which the target segment was NOT from the studied or modified program)

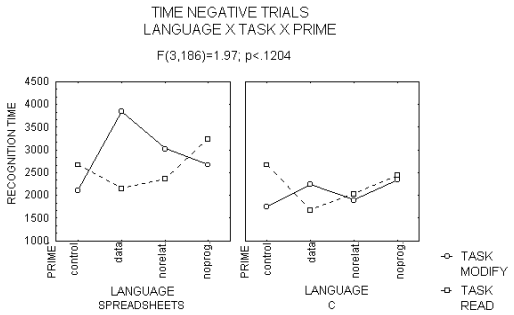

Figure 3*. Priming Recognition Results: Response time to Negative Trials, second order interaction: Programming Language by Comprehension Task and by Prime Condition.*

Recognition Time: The effect of Language was significant ($F(1,61)=14.6$; $p=0.0003$). Again, C programmers were faster (Average=2135 ms) than Spreadsheet programmers (Average =2768 ms). The interaction of Prime Condition by Comprehension Task was also significant ($F(3,186)=10.72$; $p< 0.001$). The data showed that when programmers read a program, the Control Flow condition slowed down the recognition process in comparison with the other conditions. In the opposite way, when programmers had done a modification task the Data Flow targets needed more time for recognition. It seems that programmers have acquired the Control Flow information in the reading task, and activating this information in the Control Flow condition made it more difficult to refuse the incorrect target. The same interference effect seemed to happen with the Data Flow information for the modification task.

Again the effect of this interaction seemed to be modulated by a second order interaction close to significant, of Language by Prime Condition and Comprehension task ($F(3, 186)=1.97$, $p< 0.1204$, LSD= 671). When C programmers had to read the program there was an inhibition effect of the Control Flow conditions in comparison with the Data Flow conditions. On the other hand, when C programmers modified the programs, there was a tendency for the Data Flow conditions to increase the time needed for recognition although this difference was not significant. With regard to the Spreadsheet programmers, they also performed differently depending on the task. There were no differences among prime conditions when they had to read a program. On the contrary, after modifying the programs, Data Flow and Control Flow conditions had the opposite effects. Data Flow conditions slowed down the process of rejecting a fragment, while seemed to need less time to take the correct decision.

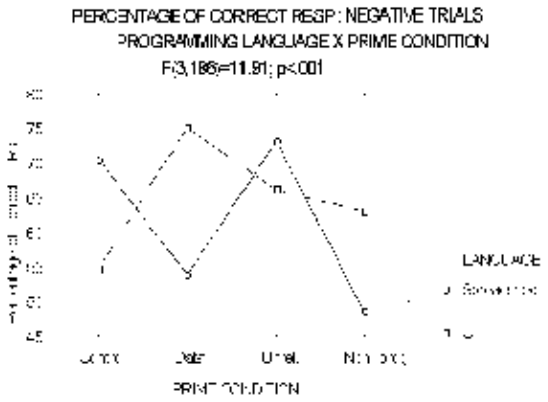

Figure 4*. Priming Recognition Results: Percentage of Correct Responds to Negative Trials, second order interaction: Programming Language by Prime Condition.*

Accuracy: There was a significant effect of Prime Condition F(3,186)=5.20; p< .002). Target segments that belonged to the studied program (Data and Control Flow conditions and Program Unrelated condition) were recognised better than targets that belonged to different programs.

The interaction of Language by Prime Condition was also significant (F(3,186)=11.91; p<0.000; M.S.E.=811.5, LSD= 10). For C programmers Control Flow conditions decreased accuracy while for Spreadsheet programmers Data Flow conditions decreased accuracy. So, Control Flow conditions inhibited recognition for C programmers and Data Flow conditions inhibited recognition for Spreadsheet programmers.

As a conclusion to all these data, we found a facilitation effect in the recognition task from our priming conditions. These facilitation effects in recognition time were not due to a trade off with accuracy as we could see in the accuracy data.

We also found strong inhibition effects both in time and accuracy. Inhibition data have been more informative than facilitation data in psychological research. In our case the interference shown specifically for the Control Flow conditions in C programmers, and for Data Flow conditions in Spreadsheet programmers gives evidence that programmers develop different mental representations based on Control Flow or Data Flow respectively.

GENERAL CONCLUSION

Our conclusions are derived from the two tests that we used to access a subject's mental representations. Our data indicated that using these methodologies together could give us complementary information which allow us a deeper understanding of both the mental representations and the processes beneath them.

In general, our results showed evidence that imagery influenced programmers' mental representations. Specifically, we found differences in the information acquired by the programmers depending upon the Language and the comprehension task. The data from the sorting task showed that the mental structure of the spreadsheet programmers has more information about both Control and Data Flow structures. However, C programmers seem to have a better mental representation for the Control Flow information. With regard to the Priming Recognition task, we found strong interaction effects when the programmers had to reject a target that was not from the program. C programmers seemed to be more influenced by the Control Flow primes, so their representation was more strongly based in Control Flow information. When Spreadsheet programmers read programs, they were more influenced by the Control Flow primes. However, after modifying the program, the Data Flow primes showed a stronger influence on recognition. These results are congruent with previous research showing differences in the program representation depending on the task and programming language (Pennington, 1987, Burkhardt, Détienne, Wiedenbeck, 1997).

Furthermore, these data together gave support to our hypothesis that the Spreadsheet, with its visual characteristics, helped programmers to develop a representation of the program based on Data Flow structure. The sorting data clearly showed that the Spreadsheet programmers develop a Data Flow mental representation even in the easiest tasks, while C programmers seemed to need to have special task conditions (e.g.- more difficult) to acquire this information. Therefore our data with C programmers replicates the results of Pennington (1987), supporting her Two Stages theory for procedural languages. Based on the effects found for Spreadsheet programmers, we claim that it is necessary to add a factor to this theory to account for the role of imagery in programming. Programmers will go through the stages that have been proposed depending on the variables that enhance the processes underlying these stages. One of these variables seems to be imagery, which enhances access to Data Flow information.

ACKNOWLEDGEMENTS:

The authors would like to thank Professor Jorma Sajaniemi and Paul Charette for all his help in the preparation of this work. The several departments and organisations that provided us the programmers for the experiment also deserve our grateful thanks.

REFERENCES

Bajo, M. T.(1988) Semantic facilitation with pictures and words. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 4, 579-589.

Burkhardt, J.M.; Détienne, F.; Wiedenbeck, S. (1997) Mental representations constructed by experts and novice in object-oriented program comprehension. In *INTERACT'97*.

Corritore, C. L.; Wiedenbeck, S. (1991) What do novices learn during program comprehension? *International Journal of Human-Computer Interaction*, 3·(2), 199-222.

Davies, S. P. (1991) The role of notation and knowledge representation in the determination of programming strategy: a framework of integrating models of programming behaviour. *Cognitive Science*, 15, 547-572.

Détienne, F. (1996) What model(s) for program understanding? UCIS'96, Pointers, France, September, 1996.

Gilmore, D. J.; Green, T.R.G. (1988) Programming plans and programming expertise. *The Quarterly Journal of Experimental Psychology*, 40A (3), 423-442.

Mayer, R. E.; Gallini, J. K.. (1990) When a illustration worth ten thousand words*? Journal of Experimental Psychology,* 4, 715-726.

Navarro, R.; Cañas, J.J.; Bajo, M.T. (1996) Pictorial aids in computer use. In T.R.G Green,.; J.J Cañas,.; C Warren,. (eds.). *Proc. of the 8 Th European Conference on Cognitive Ergonomics.* pp. 77-82. Granada.

Pennigton, N. (1987) Stimulus structures and mental representation in expert comprehension of computer programs. *Cognitive Psychology*, 19, 295-341.

Robertson, S. P.; Yu, C.C. (1990) Common cognitive representations of program code across task and languages. *International Journal of Man-Machine Studies*. 33, 343-360.

van Dijk, T.A.; Kintsch, W. (1983) Tapping into tacit programming knowledge. *IEEE Transactions on Software Engineering*, SE-10, 595-609.