The Roles Beacons Play in Comprehension for Novice and Expert Programmers

Martha E. Crosby
Department of Information and Computer Sciences
University of Hawaii, USA
crosby@hawaii.edu

Jean Scholtz
Visualization and Usability Group
NIST, USA
jean.scholtz@nist.gov

Susan Wiedenbeck
College of Information Science and Technology
Drexel University, USA
susan.wiedenbeck@drexel.edu

Keywords: POP-II.A. novice/expert, POP-II.B. program comprehension, POP-III.A. search, POP-III.C. procedural/object.

Abstract

In this research, progressively refined methods of investigating Beacon-like features (the surface features of computer programs that serve as keys to facilitate program comprehension) were used to determine how programmers from different experience levels understand a typical simple program. In this study, we found Beacons for novice programmers are different than they are for more experienced programmers. Novices discriminate very little between areas of the program and thus do not seem to use beacons. More experienced programmers tend to concentrate on the important areas of a program and thus, seem to rely on Beacons.

Introduction

The comprehension of computer programs absorbs a considerable amount of a programmer's time and effort. Programmers, who are developing software, need to be involved in the process of program comprehension in order to find errors. In this instance, programmers need to understand their own previously written code. Furthermore, maintenance tasks necessitate that programmers understand programs that may be written by other programmers. Even in learning new computer languages and algorithms, programmers use the comprehension process to comprehend example programs. An understanding of the comprehension process is necessary in order to design tools to aid programmers in this effort.

One theory of program comprehension, first discussed by Brooks (1983), involves the use of stereotypical segments of code or Beacons. Beacons are related to the role-expressiveness of a program (Green and Petre, 1996), i.e. how the notation and conventions of a program help to convey its meaning. Work by Gellenbeck and Cook (1991) has shown that Beacons in programming encompass more than just code fragments. Meaningful procedure and variable names also aid program comprehension. Possibly, comments and program structure are also involved in the process of comprehension. Hence, a plausible definition of a Beacon is a typical indicator of the programs' functionality. Brooks (1983) maintained that comprehension is a top down process in which the presence of beacons allows the programmer to make hypothesis about the function of the program. Wiedenbeck (1985, 1986) showed that there was a relationship between programmer expertise, comprehension of programs and recognition of Beacons. Wiedenbeck and Scholtz (1989) examined the relationship between the presence of a Beacon and program comprehension. The concept of

Beacons as typical indicators of program function has subsequently appeared in the software engineering literature as a partial explanation of professional programmers' facility in code recognition and comprehension (von Mayrhauser and Vans, 1995, 1996). However, the Beacons used in all of studies mentioned were obtained using expert intuition. What is needed is a way to identify Beacons by the information that the programmers really use. That is, can measurements taken to elicit a programmers' comprehension processes be used to discriminate Beacon-like and non-Beacon-like entities? If so, is this recognition of Beacon-like structures a factor of programmer expertise? Basically, do novice programmers recognise and use the same Beacon structures as more advanced programmers?

The studies described in this paper concern the identification of Beacons by programmers with different levels of expertise. Shneiderman (1980) discussed several ways to study program comprehension. We have used two of the typical methodologies, recognition and timing. However, each of the studies uses different methods of assessing what programmers consider to be indicative of the function of a program. These experiments build on each other and use a progressively refined method to investigate the identification of Beacon-like features. The first experiment asked programmers to rate lines of code as to how indicative they were of program function. The second experiment measured response time and response correctness when programmers were asked to determine which program a line of code was most likely to represent. To capture data at a very refined level, we wished to use a different, more precise methodology. Thus, the third experiment used eye tracking to precisely determine not only what programmers viewed but also how long they fixated on each type of program statement.

Eye tracking has been used to investigate reading comprehension in many studies reported in the literature. Although most of this research investigates reading of natural language text, the variety of results indicate that eye movement data could be used to successfully investigate cognitive strategies of viewing programs. Eye movements have been used in several studies as a way of exploring the participants' focus of attention (Rayner, 1978, Shebilske and Reid, 1979; Ehrlich and Rayner, 1981). The position and duration of eye fixations show which areas the subject considers important.

The three types of large eye motions are tracking, vergence movements and saccadic movements. Tracking consists of head movements; vergence movements allow the eyes to converge on the fixation point and the saccadic movements are the changes from one fixation point to another. The saccadic movements, particularly the position and duration of the eye fixation, are variables in many reading models (Gould and Schaffer, 1965; Ditchburn, 1973). There are two assumptions that link the eye fixation data to the comprehension process. The first is the immediacy assumption where the reader tries to interpret each piece of text as soon as possible. The second is the eye mind assumption which poses that the eye stays fixated on a word for as long as it takes to process the word even if comprehension involves use of information previously encountered. Just and Carpenter (1980) devised a reading model based on the duration of each word fixation. They suggest that readers are able to pace the rate they acquire information so that it matches their internal comprehension process. When readers use text that is appropriate for their reading level, their average gaze length is 1.2 words. Additional studies by Carpenter and Daneman (1981) and Just and Carpenter (1978) support the view that readers pause on words that require more processing. Subject differences were found in areas other than reading. Vonder Embse (1987) used eye movements to demonstrate that expert and novice students view mathematical graphs differently. The experts focus on the meaningful part of the graph while the novices attend equally to all parts of the graph. The immediacy and eye-mind assumptions differ from the models of Bouma and deVoogd (1973) and Bouma (1974) which propose an internal buffer. Crosby and Stelovsky (1989, 1990) showed that programs are read differently than prose and that complex material such as programs support the immediacy and eye-mind assumptions. Results from these studies suggest that eye tracking could indicate what portions of the program are considered important and thus facilitate the understanding of program comprehension.

Experiment 1

Forty-five programmers from the University of Nebraska-Lincoln and the University of Nebraska-Omaha computer science department participated in experiment 1. These students were classified as novice programmers, intermediate programmers or advanced programmers. The novice programmers were selected from computer science (CS) students who had completed or were just completing a second term of Computer Science II. Intermediate programmers were selected from CS students at the junior or senior undergraduate level. In addition to Computer Science I and II, these students had all completed a course on algorithms. Most had also completed several other courses that required significant programming: assembly language, programming languages, UNIX programming, and operating systems. The advanced programming group comprised graduate CS students and CS faculty. The members of this group were chosen because they had high experience programming professionally or as part of their research. Some members of this group also taught programming. The materials consisted of listings of different programs that were administered to the groups of programmers both at the University of Nebraska-Omaha and the Human Computer Interaction (HCI) laboratory at the University of Nebraska-Lincoln. The lines of code presented to programmers were based upon a typical binary search program. These lines of code were augmented by mnemonic versions of lines 4 and 5 that are shown in Figure 1 (Lines L7 and L8).

The programmers were given a packet of materials that contained lines of code from the binary search program in random order. The code was further augmented with mnemonic lines and distracter lines of code from a greatest common divisor program. The programmers were then asked to rank (on a 6 point scale) how likely it was that a particular line of code came from a binary search program. The rating scales ranged from "very unlikely" to "very likely." If the participants judged the line of code to be "very unlikely" it received a score of 1. If the participants judged the line of code to be "very likely" it received a score of 6.

Line id	line contents
L1	function binarysearch (var a: atype; v,n: integer): integer;
L2	var x,y,m: integer;
L3	y :=n;
L4	m := (x + y) div 2;
L5	if v < a[m] then y := m - 1 else x := m + 1;
L6	if v = a[m] then binarysearch := m else binarysearch := 0;
L7	mid := (high + low) div 2;
L8	if x < a[mid] then high := mid - 1 else low := m + 1;

Figure 1: The binary search code used as the basis for experiment 1.

Results

An ANOVA of the rating data showed that expertise was significant, $F(2,336) = 3.37$, $p \leq .05$. The Newman-Keuls specific comparison test showed that the intermediate group of participants was significantly different from the advanced group of participants. The novice programmers, however, interacted with both groups. The results were also significant with respect to the lines being rated, $F(7,336)=4.82$, $p \leq .0001$. A Newman-Keuls test for specific means showed that line L1 was significantly different from lines L3 and L5. Line L1 was the most indicative of the binary search algorithm while lines L3 and L5 were the least indicative of this algorithm. Recall that L1 is the function header, L3 is an assignment of a variable and L5 is the non-mnemonic version of resetting the limits. The fact that the lines L7 and L8 are rated higher than L4 and L5 show the effect of mnemonic names on comprehension. The interaction between the lines and the levels of expertise, however, was not significant. The results suggest that the novices did not discriminate between the lines as well as the intermediate and advanced programmers. L6, L4 and L5 have the greatest difference in the means for the three groups of programmers. An ANOVA showed that only the lines L4 and L5 were significant with respect to levels of expertise. For both lines L4 and L5 the mean rating for the advanced subjects was lower than the mean rating for the intermediates. These lines were the non-mnemonic versions of resetting limits for the search and calculating a new mid value for

the search. A possible explanation for the advanced group's lower rating is the programmers' realisation that these two lines of code as written could also appear in other types of programs. The calculation of the midpoint value could be used in any situation where the integer mean of 2 values is required. The advanced programmers, having been exposed to a wider variety of programs and algorithms, tend to be even more discriminating than the intermediates. Novice programmers were much less able to discriminate among the different lines, yet he novice programmers were grouped with the advanced programmers for line L4 and with the intermediate programmers for line L5. The other lines in the program, such as the header L1 and the variable declaration L2 were more closely agreed upon as being more indicative in the case of L1 and less in the case of L2. A specific comparison analysis (Student-Newman-Keuls test) showed that no significant difference between the ratings given the lines for the intermediate and the novice programmers. The mean rating of the advanced programmers for line L5 was significantly lower than it was for the rest of the lines.

For the binary search program we found that expertise and the lines being rated contributed to the comprehension measure. Since the means were always close for at least two groups although the composition of these two groups varied there was no interaction effect. Unexpectedly, novice programmers sometimes agreed with the intermediate programmers and sometimes with the more advanced programmers. The mean ratings for the lines were only statistically significant for the advanced subjects. It appears that less experienced programmers are unable to distinguish which lines of code are indicative of appearing in a certain type of program but that the more advanced programmers do make these distinctions. Although this study was exploratory, the results suggest that more advanced programmers seem to use different Beacons to facilitate program comprehension than programmers with less experience. Experiment 2 was designed to further test this possibility.

Experiment 2

Thirty programmers from the University of Nebraska-Lincoln and Portland State University participated in experiment 2. Fifteen of these subjects were intermediate programmers and fifteen were advanced programmers. Intermediate programmers were selected from CS students at the junior or senior undergraduate level. The advanced programming group comprised graduate CS students and CS faculty. The programming background of these two groups was the same as participants in experiment 1. Novice programmers were not used as the experiment as they do not appear to be able to reliably recognise beacons. In experiment 2, lines of code from the binary search program were presented to subjects on a computer monitor. As distracters lines of code from a depth first search and a shell sort were also presented. A personal computer was used to present the lines of code and to record the reaction time as well as the response. A program presented the lines (in random order), sorted the response times and sorted the responses, making a file for each participant. Half of the experiments were conducted in the HCI laboratory at Portland State; the other half were run in the HCI laboratory at the University of Nebraska - Lincoln. The participants were presented with lines of code from a binary search, a depth first search and a shell sort. They were then asked to press a key to indicate which program produced each line of code. The answer and the response time were recorded for each line of code presented.

Results

In order to compare the results from experiment 2 to those from experiment 1, analysis was done on the data from the binary search algorithm. The independent variables were expertise and line number and the dependent variables were response correctness and response time. The response was classified as incorrect if the programmer answered either depth first search or shell sort instead of binary search. The percentage of correct responses was not significantly different for the two groups of programmers. For the lines of code belonging to the binary search program, 63 % of all the intermediates' responses were correct while 66% of the advanced groups' responses were correct. The mean reaction time for the advanced group was, however, significantly lower than that of the intermediates. $F(1,168) = 16.62$, $p \leq .0001$. The mean reaction time was not significantly different between the lines of code. But the percentage of correct responses differed significantly for the given lines, $F(5,168) = 9.50$, $p \leq .0001$. Although the intermediate programmers were able to determine

which lines came from the binary search program as well as the advanced programmers, they were significantly slower in doing so.

The correctness measure for the intermediates differed on the different lines of code. A comparison of the response correctness by lines, showed that lines containing the heading and the function value return were significantly different from the other four lines. This corresponds to the results in experiment 1. That is, the same two lines have the highest ratings, however, only the rating for the heading line is statistically significant. Figure 2 shows the mean response time for each line by expertise. An ANOVA using the correctness of response as the dependent variable showed that expertise was statistically significant for the responses. In each case, however, the advanced group completed the task in less time. For correct responses, $F(1,168)$ equalled 8.37, $p \leq .05$ and for incorrect responses $F(1,168)$ equalled 10.88, $p \leq .05$. The line was more of a factor for subjects with incorrect responses, $F(5,168) = 1.75$, than for subjects giving a correct response, $F(5,168)=0.33$, but neither case was statistically significant.

Interestingly, the advanced programmers scored 100% in identifying L6 as having come from a binary search program while in experiment 1, advanced programmers only gave a mean rating of 4.7/6.0 to this line. Possibly the programmers in experiment 2 had an easier decision since they were given a more structured choice. Both the intermediate and advanced programmers may be very familiar with the binary search program and therefore use the same lines as beacons. The intermediate and more advanced programmers displayed the same ability in terms of percentage of correct responses. The more advanced programmers were able to accomplish this in significantly less time, regardless of the line of code. The lines that were more indicative of a binary search program, as judged by correctness, were the heading and the return value. These results agree with those from experiment 1, that is, these two lines had the highest ratings.

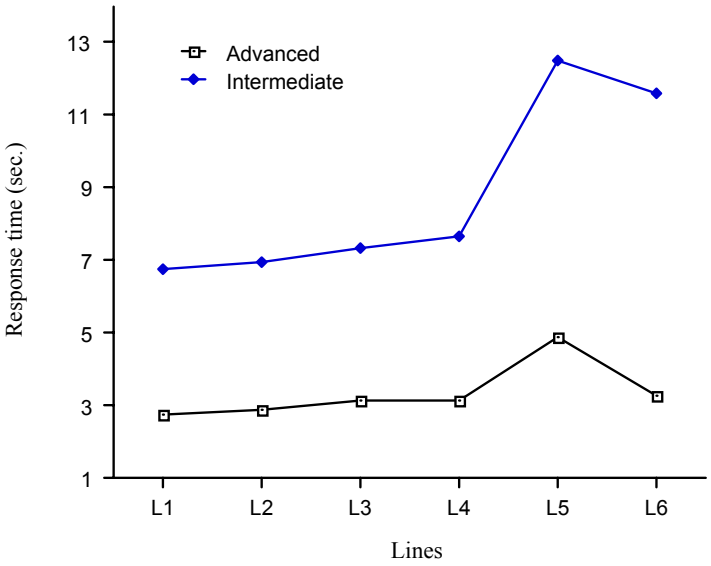

Figure 2: Mean Response Time by Line and by Experience Group.

Experiment 3

Nineteen programmers were randomly selected from the University of Hawaii's computer science program. For this experiment, the novice experience group (LOW) consisted of nine students with one semester of programming experience. The advanced group (HIGH) consisted of CS faculty members and students from advanced undergraduate and graduate classes. These participants had at least four years of programming experience.

The experiment was conducted at the eye-tracking laboratory at the University of Hawaii. The equipment used consisted of an ASL eye-movement monitor, controlled by a host computer. The Eye

Monitoring System has digital output that samples the eye position and pupil size 60 times every second. The host computer was used 1) to prepare scenes to be presented to the subject in experiments, 2) to present scenes to the subject and to control the sequencing of the experiment, 3) to collect data during the experiment, and 4) to examine and analyse the data collected in experiments.

A slightly more complex implementation of the binary search program was selected but the lines were matched to those used in the first two experiments. Comprehension tests were given before and after the experiment. The pre-test asked general questions about the concept of the binary search and the post-test asked both general and specific questions about how the algorithm worked. In addition, each participant completed a Cloze (fill in the blanks) test before and after the presentation of the algorithm. The participants were tested individually. First, they took the pre-test. Then the binary search program was presented. The participants were instructed to read the algorithm for comprehension and to try to remember it. When the participants finished reading the algorithm to their satisfaction, the algorithm was removed and the participants were given a comprehension post-test and a Cloze test.

The dependent variables were the number of fixations and fixation times. The number of fixations and the fixation times were highly correlated ($p \leq .0001$). We are reporting the analysis based on the number of fixations, but the results using fixation times are highly correlated. We examined the total number of fixations for all participants. Based on clusters of fixations, we partitioned areas of concentration into classes (Figure 3 shows these partitions). These partitions, which contained different amounts and types of information, were classified into the following categories: comments, comparisons, complex statements, simple statements and key words. Since we were interested in how the programmers used the categories as beacons, we combined the categories throughout the program (converted to percent to account for participants' different viewing times).

Each of the areas offered different information about the type of knowledge elicited from the program. Although the "comments" indicate the program's purpose, statements, comparisons and their interrelations are indispensable for detailed comprehension. The "comparison" areas are deceptive. Although they are seemingly easy to understand, it is hard to recall the direction of the comparison operator. The "complex statements" provide the greatest amount of information. "Simple assignments" are statements that are moderately complex, but essential to the comprehension of the program, as they provide the necessary contextual information. "Key words," such as "BEGIN" and "END," are predictable and provide almost no information; thus can be considered the least important area of the program. These categories were related to the statements in experiment 1 and experiment 2. "Comments" corresponded to L1 and L6, as the word binary search was included in the text," key words were related to L2, "simple assignments" to L3, "comparisons" to L5 and L8; and "complex statements" to L4 and L7.

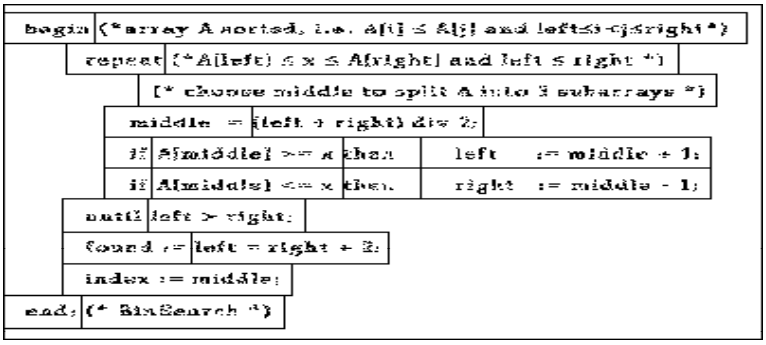

Figure 3: Subdivision of the algorithm text into areas.

Results

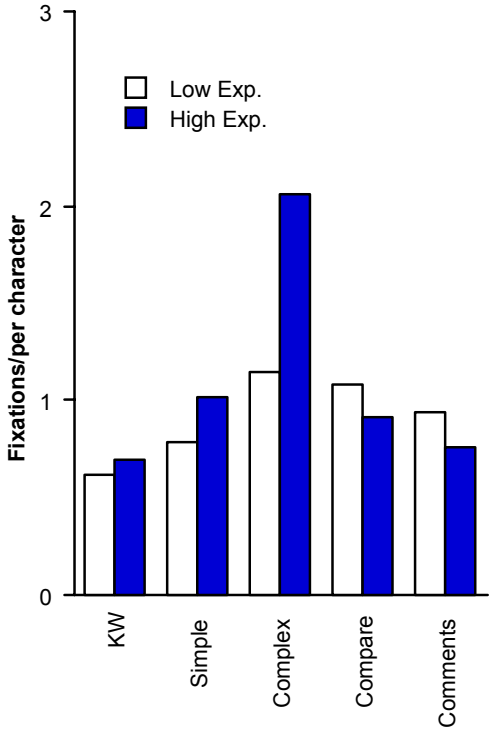

Figure 4: Percentage of fixations in areas by programmer experience.

As we found in experiment 1, the program areas were viewed differently depending on the participants' experience. The category with the largest number of fixations was the comments. However this was also the largest area. Since fixations increase with length, we needed to adjust for the different lengths in each category to address this question. We adjusted the number of fixations by dividing by the number of tokens (words and symbols) in each category. We expected the complex statement category to be the beacons. As Figure 4 illustrates, complex statements were the most influential category for the high experience group.

The variant of binary search used in the experiment was unusual. In particular, the complex Boolean assignment "found := left = right + 2" distinguishes this version from more common implementations. The high experience group had general knowledge about several variations of the algorithm. We hypothesised that the more complex the material becomes, the more attention it would receive from the high experience and high comprehension groups. To test this conjecture, we analysed the number of fixations in the area "left = right + 2". The results confirmed this hypothesis, the average fixation duration was statistically significant for experience (R^2 = 21% and $p \leq .01$) and comprehension (R^2 = 38% and $p \leq .001$). Since keywords are predictable, it was not surprising that they did not attract much attention. The viewing of comparisons yielded significant differences between the experience groups.

Discussion

The binary search was used as an example of a typical program familiar to computer science subjects, yet with enough variations that understanding is not automatic. Yet, everyone does not recognise similar salient features of the program. This finding suggests that "Beacons may be in the eye of the beholder." Experiment 1 showed that intermediate and advanced subjects identified meaningful Beacons but novices did not. Experiment 2 indicated that both intermediate and advanced programmers could recognise lines that belonged to the binary search, although the advanced subjects performed the task faster. Experiment three found that advanced programmers used complex

statements as beacons while novices made little distinction between the different classes of program statements.

Prior studies have suggested that experience level is a key factor in processing information about programs and our experiments have confirm these results. Certainly, programmers who understand the algorithm have had to rely on some part of the text to facilitate their understanding. Our studies have provided more detail about what the programmers really use as Beacons, or at least where they put their focus of attention. Analysis revealed that for experienced programmers the complex statements were fixated the longest and were likely to be Beacons. The high comprehension participants also fixated proportionally longer on the comparisons. Key words and simple statements did not appear to be the Beacons for those participants who understood the program. Programmers, who are experienced or understand the material, view information selectively and focus on the more complex parts of the information.

These experiments show that for more experienced programmers, portions of the code differ in the amount of information that can be extracted. This research suggests that there are two primary types of Beacons that appear in programs: comment Beacons or lines of code that contain a mnemonic hint about program functionality and more complex Beacons which contain the key or focal line of the program. Results from these experiments suggest that comment beacons are very indicative of program functionality and can be processed quickly by the more experienced programmers. Complex and comparison Beacons, however, consume considerably more comprehension time but are also indicative of program functionality. It would seem that aids to comprehension should address complex and comparison type statements. A question which we intend to address in future research is how the inclusion of in-line comments concerning complex and comparison statements affect the time to comprehend these statements. Results from this research suggest that mnemonic names could speed up comprehension time. Besides the time element in comprehension, we intend to address the element of sequence to see in what order programmers tend to examine portions of the code. If programmers adhere to Brook's method of hypothesis and hypothesis refinement, the sequence in which portions of code are examined is a determining factor in hypothesis generation. This research is summative, that is it only deals with the total pattern of the performance of programmers from different experience groups. In future studies we need to examine the “process” of selecting beacons. Individual differences in recognition, reaction time and eye tracking can be invaluable in finding out more about the how beacons are selected. Research by Crosby and Stelovsky (1989) suggests there are a wide variety of viewing strategies when subjects read programs. Will all of these strategies allow users to use beacons to their advantage? Future studies will address this question.

This area of research may have implications for computer interface designers. If a user views screen area with complex material, such as an algorithm, in a cursory way, it is unlikely that this information is absorbed. This information can be particularly useful if a wealth of information needs to be presented simultaneously on the screen. Applications could be built to detect and highlight critical, but unattended areas. Moreover, more redundancy, additional help or alternate readings can be offered if it is unlikely that the information was sufficiently digested.

Acknowledgements

This first author was supported by ONR grant no. N00014970578 and DARPA grant NBCH1020004. The third author was supported by a grant from the National Science Foundation, Program on Information, Robotics, and Intelligent Systems.

References

Bouma, H., (1973) Visual interference in the parafoveal recognition of initial and final letters of words. *Vision Research*, **13**, 767-782

Bouma, H., and deVoogd, A. (1974) On the control of eye saccade in reading. *Vision Research*, **14**, 273-284

Brooks, R., (1983) Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies*, **18**, 543-554.

Carpenter, P. and Daneman, M. (1981) Lexical access and error recovery in reading: A model based on eye fixations. *Journal of Verbal Learning and Verbal Behavior*, **20**, 137-160.

Crosby, M. and Stelovsky, J. (1989) Subject differences in the reading of computer algorithms. In *Designing and Using Human-Computer Interfaces and Knowledge Based Systems,* G. Salvendy and M. Smith Eds., Elsevier Science, Amsterdam, 137-144.

Crosby, M. and Stelovsky, J. (1990), How do we read algorithms? A case study. *IEEE Computer*, **23**, (1) 24-35.

Ditchburn, R. (1973) *Eye movements and visual perception*. Oxford: Clarendon.

Ehrlich, S. and Rayner, K. (1981) Contextual effects on word perception and eye movements during reading. *Journal of Verbal Learning and Verbal Behavior*, **20**, 641-655.

Gellenbeck, E. and Cook, C. (1991) An Investigation of Procedure and Variable Names as Beacons During Program Comprehension. (Tech Report No. 91-60-2). Corvallis: Oregon State University, Computer Science Department.

Gould, S.D. and Schaffer A., (1965) Eye movement pattern in scanning numeric displays *Perceptual and Motor Skills*, **20**, 54-335.

Green, T.R.G. and Petre, M. (1996). Usability Analysis of visual programming environments: A 'Cognitive Dimensions' framework. Journal of Visual Languages and Computing, 7, 131 - 174

Just, M. and Carpenter, P. (1978) Inference processes during reading: Reflections from eye fixations. In J. Senders, D. Fisher, and R. Monty (Eds.), *Eye movements and higher psychological functions,* Hillsdale, N.J.: Erlbaum.

Just, M. and Carpenter, P. (1980) A theory of reading: From eye fixation to comprehension. *Psychological Review*, **87**, 329- 354.

Rayner, K. (1978) Eye movements in reading and information processing. *Psychological Bulletin*, **85**, 618-660.

Shebilske, W. and Reid, L. (1979) Reading Eye Movements, Macrostructure and Comprehension Processes. In: P. Kolers, M. Wrolstad and H. Bouma Eds: *Processing of Visible Language 1,* Plenum Press, New York.

Shneiderman, B. (1980) *Software Psychology*. Winthrop, Cambridge, MA. 13-29.

Von Mayrhauser, A. and Vans, A.M. (1996). Identification of dynamic comprehension processes during large scale maintenance. IEEE Transactions on Software Engineering, 22(6), 424-437.

Von Mayrhauser, A. and Vans, A.M. (1995). Program comprehension during software maintenance and evolution. *Computer*, 28(8), 44-55.

Vonder Embse, C. (1987) An eye fixation study of time factors comparing experts and novices when reading and interpreting mathematical graphs. *UMI Dissertation #8717747*, Ohio State University.

Wiedenbeck, S. (1986) Beacons in computer program comprehension. *International Journal of Man-Machine Studies*, **25**, 697-709.

Wiedenbeck, S. (1985). Novice/expert differences in programming skills. *International Journal of Man-Machine Studies*, 23, 383-390.

Wiedenbeck, S. and Scholtz, J. (1989) Beacons: A knowledge structure in program comprehension. In G. Salvendy and M.J. Smith (Eds.), *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*. Elsevier, Amsterdam, The Netherlands, 82-87.