An Experiment on the Effects of Program Code Highlighting on Visual Search for Local Patterns

Tuomas Hakala, Pekka Nykyri, and Jorma Sajaniemi

University of Joensuu, Department of Computer Science,
P.O.Box 111, 80101 Joensuu, Finland,
{thakala|pnykyri|saja}@cs.joensuu.fi

Abstract. Many current program editors use syntax highlighting but effects of various coloring schemes are not known. This paper presents the results of an experiment where three coloring schemes were used by intermediate programmers in visual search tasks for local patterns in Java programs. Differences between the coloring schemes were small and not statistically significant. Especially, in contrast to intuition, the control scheme, black text on white background, resulted in the same overall search performance as the other coloring schemes. Differences between search target types were statistically significant and each coloring scheme turned out to be best for some target type but the interaction was statistically only almost significant.

1 Introduction

A program editor is an important tool in many programming tasks. It is not only used for writing a program but also in program comprehension, in debugging etc. Even the process of writing a program includes episodes of program comprehension and recall of details written earlier [1]. These tasks can be enhanced with various visualization techniques, e.g., coloring automatically different parts of the program, popping up dynamically explanations of constructs as the user moves the mouse over a construct etc.

It seems obvious that different tasks benefit from different techniques. Consider, for example, the use of some special color for reserved words. In writing a program, it gives immediate feedback for misspelled words as the programmer unconsciously monitors the word to see if it changes its color when finished. On the other hand, when trying to comprehend a program, individual keywords do not correspond to meaningful structures and their coloring may thus be of dubious value. Thus any study of the effects of program code visualization must clearly state the specific programming tasks that the study intends to cover.

Many current editors use syntax highlighting at the level of *lexical tokens*, i.e., keywords are highlighted with one color, literals in another color, variables in a third color etc. Coloring can also be based on *larger syntactical constructs*, for example by giving some color to all declarations, another to all loops etc [2]. A coloring scheme can be based on non-syntactic constructs also, for example on the *plan structure* of the program [3].

From a cognitive point of view, the coloring of programming plans [4] seems to be closest to programmers' mental representations of programs. Indeed, Gilmore and Green [3] found that highlighting plans helped Pascal programmers to detect plan bugs, e.g, omitted update or failure to check for valid input. However, no improvement was found in other bug types nor among Basic programmers. Current syntax highlighting schemes are much harder to justify with programmers' cognitive structures. These schemes highlight similarity of low-level syntactical items which has never been found to be a central part of experts' knowledge structures. Even at the low programming knowledge level that consists of programming language knowledge, expert programmers have been found to group keywords according to the common structures of the language in question [5] rather than perceive them as a single unstructured group.

Larger syntactical constructs—declarations, loops, etc—correspond better to programmers' cognitive structures but we do not know of any studies of the effects of highlighting such constructs. Neither do we know of studies of the effects of low-level lexical token coloring even though this type of syntax highlighting is common in current editors—presumably because it is relatively easy to implement and corresponds to the intuition of the designers of the editors. In this paper, we study the effects of these types of program code highlighting with respect to the task of visually searching for local patterns.

We have selected visual search as the basic task because it occurs as a subtask in many programming activities like program comprehension and error detection. The targets are local patterns, i.e., code passages that can be identified by looking at a few consecutive program lines because we wanted to minimize the effects of individual scan order preferences caused by personal program comprehension strategies. As a result the tasks represent elementary operations in many programming activities that are accomplished with a program editor. There are, however, programming activities where visual search for local patterns is only a minor part, e.g., writing new program code that has been designed carefully before the actual coding. Such tasks are not covered by our study.

The rest of this paper is organized as follows. The next section provides background with a literature review. Section 3 describes the experiment and presents its results. Section 4 contains a discussion of the results and, finally, Section 5 contains the conclusion.

2 Related Literature

This section contains a review of related literature on visual search and on proposed ways to use color for program code highlighting.

2.1 Visual Search

Visual search has been studied extensively in cognitive psychology (see, e.g., [6] for a review). Desimone and Duncan [7] present two types of basic phenomenon in visual attention: limited capacity for processing information, and selectivity.

Limited capacity means that dividing visual attention between two objects results in poorer performance than focusing attention on one—independently of the complexity of the properties to be searched for. Moreover, this interference is largely independent of the spatial separation between the objects.

Selectivity concerns the ability to filter out unwanted information. In easy cases the target is clearly different from rest of the information and pops out very quickly. In these cases, the number of nontargets has little effect on the speed or accuracy. For example, a target with some specific visual characteristic is easy to find if the nontargets are homogeneous and lack this visual characteristic, e.g., a colored target in a multicolor display may show good pop-out if the colors are highly discriminable. Easy cases are characterized by a bottom-up, or stimulus-driven, bias.

In hard cases, nontargets are similar to the target and do not filter out effectively. Targets may have a complex structure and searches are guided by a top-down, or goal-directed, control. This control is based on an "attentional template" that forms the short-term description of the target and can specify any properties of required visual input. However, even in these cases, the ability to find targets is still dependent on bottom-up stimulus factors, especially the visual similarity of targets to nontargets.

Finally, Desimone and Duncan [7] point out that search can furthermore be guided by some cue to the location of the target; or by bias derived from long-term memory, e.g., novel targets are easier to find than familiar forms, and targets that have long-term learned importance are hard to ignore even if they are non-important in the current search.

Itti and Koch [8] note that depending on the context some stimuli can be very salient. If the stimuli is salient enough, so called singleton, it will involuntarily attract attention and make a pop-out effect. Thus salience is determined very fast in a pre-attentive manner across the entire visual field. On the other hand, top-down search that is based on an attentional template requires voluntary effort that exceeds the time needed to move the eyes.

Egeth and Yantis [6] review studies that have come to different conclusions about singletons capturing attention. A possible explanation is that the different findings originate from two different attentional strategies that people adopt. In some circumstances, people enter a "singleton detection mode" and in some other conditions they enter a "feature search mode". In the singleton detection mode, attention is directed to the largest local feature contrast, and in the feature search mode to locations that match some task-defined visual feature. Irrelevant feature singletons capture attention only in the singleton detection mode.

2.2 Color Highlighting of Programs

Gilmore and Green [3] have studied the effects of indentation and color highlighting on intermediate Pascal and Basic programmers' bug detection. They used color to cue plan [4] structure where the plans were an input (with a guard), a maximum, an average, and a filter. The cues used a different color for each plan and no more than three plans occurred together. Indentation had a more positive

effect on bug detection than plan highlighting, and it improved the detection of control bugs in both Pascal and Basic programmers. Only Pascal programmers benefited from the coloring of plan structures, and mostly in the case of plan bugs. Gilmore and Green conclude that the lack of help for Basic programmers is due to the less important role of plans in Basic programming.

Highlighting plan information is troublesome for several reasons. First, it is not known exactly what plans programmers have and plans may even vary from person to person. This makes automatic detection of plans hard and consequently their automatic highlighting unattractive. Second, as the program lines corresponding to a single plan are dispersed around the program, the screen becomes cluttered with lines of different colors heavily intermixed. Third, some parts of a program may belong to several plans making the design of coloring schemes hard and the screens even more cluttered. Finally, plan visualizations are new to programmers and even the concept of a plan is usually new to most programmers even though their tacit knowledge may contain plans. This makes the use of plan highlighting problematic in an experimental setting: the participants may just not understand what the purpose of the highlighting is.

Cigas [2] has suggested a coloring scheme to highlight the structure of Pascal programs. Structures to be colored are basically consecutive program lines that will be executed together, i.e., statements with no branches in between, and declarations. Cigas lists a set of requirements for a coloring scheme. First, each structure type must have its own color so that the type (loop, conditional, ...) can be identified by its color as well as by its keyword. Second, nested structures of the same type (e.g., nested loops) must have different colors so that they can be distinguished by the color. However, these colors must be similar so that the type (e.g., loop) can still be identified. Third, a user must be able to tell the difference between two colors, i.e., the colors must be distinctive.

Cigas suggests a specific coloring scheme that uses different colors for the following structures: the main program (red), procedures and functions (magenta), declaration sections (cyan), loops (blue), conditionals (green), and comments (yellow). Nested structures are colored in two ways: if the inner structure is different from the enclosing structure, it will have its own color (as listed above). On the other hand, if the inner structure is of the same type, then color is shifted a little to make a distinction. The background color is supposed to be black.

The coloring scheme suggested by Cigas is less cluttered than that of Gilmore and Green. Moreover, it is easier to understand by programmers because it is based on well-defined syntactic constructs that every programmer must be aware of. It is, however, an open question to what extent the fine grained coloring scheme does support programmers' cognitive processes as Cigas presents no evaluation of the effects of his coloring scheme.

The current standard of highlighting lexical tokens leads to the most cluttered screens, because each word may be colored differently from the adjoining words. Different token coloring schemes exist and users may define their own schemes. The amount of clutter may vary between different coloring schemes as some schemes use the same "neutral" color for several token types leading to less

clutter. We do not know of any scientifically robust analysis of the effects of various coloring schemes.

3 Experiment

In order to study the effects of various color highlighting techniques we conducted an experiment where participants made visual searches for local patterns in program code. We selected visual search as the basic task because it occurs as a subtask in many programming activities like program comprehension and error detection. The targets were local patterns, i.e., program constructs that can be identified by looking at a few consecutive lines and thus the tasks represent elementary operations in many programming activities. There were three highlighting conditions and three search target types.

3.1 Method

Participants: There were twenty-one participants, all of them students in the third to fourth year computer science course on software metrics at the University of Joensuu, Finland. Three of the participants were female and eighteen male. Participation was a course requirement but the participants got a small reward in the form of course credits.

Materials: The experimental materials consisted of screen captures of 12 Java program fragments with an associated instruction to search for some specific local pattern (see Figure 1). The length of the program fragments was 76 lines and they were taken from an existing software found in the web [9]. The tasks were presented using 1280*1024 resolution 17 inch TFT displays with font size of 10 pixels resulting in letters being 3.0 millimeters high on the screen.

There were three types of targets, i.e., three forms of patterns to be searched for: *assignment targets* were assignments of some specific form (including a certain operator or no operator at all); *parameter targets* were method declarations or calls with a certain number of parameters; and *statement targets* were control structures with some specific condition to look for.

The phrasing of the tasks was carefully designed not to contain any of the strings belonging to the target of the search. For example, if the target involved a multiplication operator "*", the task instruction did not contain this operator but only its name "multiplication". Thus a participant had to transform the verbal task instruction into a visual image as a part of the search. The rationale for this decision was to obtain a better validity with actual search tasks in programming where the target of a search is usually a mental pattern rather than a direct visual pattern. For the same reason, the font used for the task instruction was different from that of the program fragments.

For each task three versions with a different coloring scheme were made. The *control scheme* used black text on white background. In the *block scheme* comments were colored blue, and method headings and all declarations in red. The *token scheme* (see Figure 1) used standard Java coloring scheme of the vim

Fig. 1. User interface used in the experiment. The task instruction on the left asks for "the name of the variable that is affected by addition" in the program code fragment on the right. The answer is "mod" because of the assignment "mod += attackerMod;" on the ninth line from the bottom.

editor [10]: comments mostly in blue with some enhancements in light blue and yellow, keywords in green (in declarations) or brown (elsewhere), and literals in red. Thus the token scheme highlighted lexical tokens and the block scheme highlighted larger syntactical structures (but was less cluttered than the scheme suggested by Cigas [2]).

Both the block scheme and the token scheme used blue for comments; thus a positive transfer effect can be expected. The block scheme used red for declaration blocks whereas the token scheme used green for keywords in declarations and red for literals resulting in a negative transfer effect.

In the three differently colored versions of a task, the task instructions were the same. The program fragments were also the same except that variable and method names were changed. The names used in the three versions were similar in appearance, e.g., "getPosition", "getLocation", and "getSetting"; or "i", "j", and "k".

Design: The experiment was a within-subjects design with two within-subjects factors: coloring scheme and target type.

Each participant was presented with all 12 tasks with all three coloring schemes—one coloring scheme at a time. The order of the coloring schemes was counterbalanced.

In order to minimize learning effects, the task versions were different in the different coloring schemes, i.e., used different variable and method names. The order of task versions within a coloring scheme was random but the same to all participants in the first, second and third scheme. Thus the sequence of all tasks was the same for all participants; only the order of coloring schemes was varied.

Procedure: Participants were first presented with three practicing tasks, one using each coloring scheme, in order to get them used to the user interface. Then, for each coloring scheme, the 12 experimental tasks were preceded by two extra tasks whose purpose was to help participants to get accustomed to the new coloring scheme. No explanations of the principles used in the coloring schemes were given to the participants.

The task of participants was to read the task instructions, find the answer using a visual search, and hit Enter in the keyboard. The program fragment was then replaced by a field for the participant to enter his or her answer. By hitting Enter again, the next task was presented. The time used for reading the instructions and finding the answer (up to the first Enter) was measured. If a participant did not find an answer in the time limit of 60 seconds, the program fragment was automatically replaced by the answering field. Participants were instructed to enter in such cases the fact that the time had expired as their answer. Similarly, they were instructed to note in the answer if they did not understand a task instruction. The last task was followed by a screen asking for demographic data.

The answers and search times were recorded automatically, and correctness was decided later manually so that evident typing errors were not counted as errors.

Participants were run in two groups. After the first practicing tasks participants worked at their own pace. The lengths of the sessions were 31 and 32 minutes.

The materials and the procedure were pretested with two participants and the one minute time limit was introduced and one task with an ambiguous answer was changed as a result of this pretest. We also asked the pretest participants whether they noticed that the same search target locations reappeared in all coloring schemes and they reported not having noticed the recurrences.

3.2 Results

Among the demographic data questions, participants were asked if they had any deficiency with color vision. Three participants reported a deficiency and were not included in the analysis. Moreover, one participant reported of experiencing migraine during the experiment and was not included in the analysis. Finally, one participant reported of being able to predict the location of search targets when seeing later versions of the same task. Also this participant was discarded. Thus the data of sixteen participants was used in the analysis.

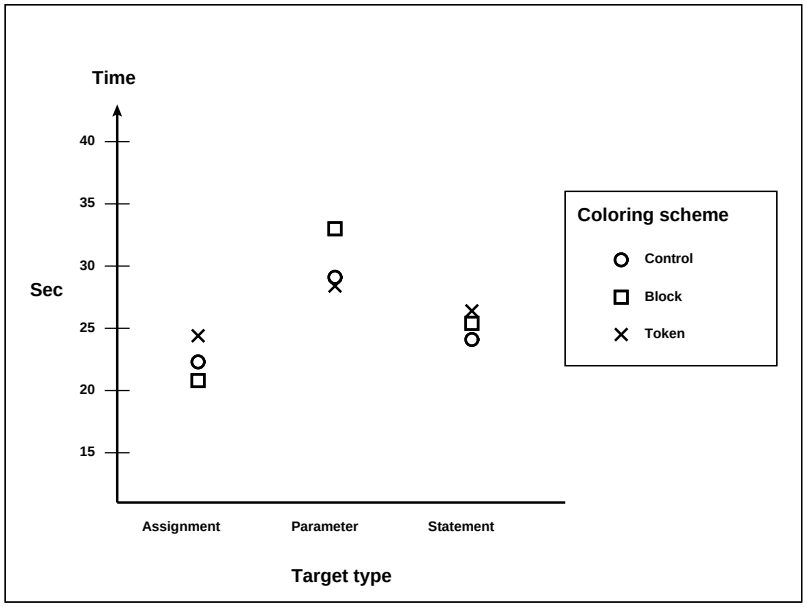

Fig. 2. Mean search times (in seconds) in the three search target types for the three coloring schemes.

Incorrect answers were checked by hand and counted as correct if it was evident that the participant had located the target, i.e., even if there were typing errors in the answer. If a participant indicated in the answer that he or she had not understood the task, the answer was not included in the analysis.

Search time: Table 1 contains the mean search times broken down by the three search target types and the three coloring schemes. The means are also depicted in Figure 2. A two-way within-subjects ANOVA on search times showed that there was a significant main effect of target type ($F(2, 30) = 11.327, p < .001$) A post-test with Bonferroni adjustment indicated that this was due to the higher search time of parameter targets as compared with assignment and statement targets ($p < .02$). The two-way within-subjects ANOVA did not indi-

Table 1. Mean search times (in seconds) and standard deviations in different target types and coloring schemes.

	Target Type					
	Assignment		Parameter		Statement	
	Mean	S.D.	Mean	S.D.	Mean	S.D.
Control scheme	22.3	12.8	29.1	7.9	24.1	9.6
Block scheme	20.8	8.8	33.0	10.5	25.4	7.3
Token scheme	24.4	12.5	28.4	7.4	26.4	12.4

Table 2. Mean search time (in seconds) broken down by the presentation order of the different coloring schemes.

Order of presentation	Mean	S.D.
First coloring scheme	33.5	16.4
Second coloring scheme	24.3	15.0
Third coloring scheme	20.2	12.8

cate main effect of coloring scheme but there was an almost significant two-way interaction of coloring scheme and target type ($F(4, 60) = 2.457, p = .055$).

The lack of statistically significant differences between coloring schemes was at least partially affected by the steep learning curve of the participants. This is demonstrated in Table 2 that gives mean search times in the first, second and third coloring scheme presented to a participant. As seen in the table, there is 40% improvement from the first to the third scheme which results in a high variance in the search times within the experimental conditions and weakens thus the results of statistical analyses.

To study the learning effect in more detail, a three-way between-subjects ANOVA using the presentation order as the third factor was performed. It showed a significant main effect of the order of presentation ($F(2, 117) = 33.074, p < .001$) and target type ($F(2, 117) = 11.084, p < .001$) but no interaction effects were now found.

To compensate for the learning effect we looked at each task in the order the tasks were presented to the participants. Remember that the first task was the same to all participants—only the coloring scheme was varied, so was the second etc. We calculated for each of the 36 tasks mean search times in each of the three coloring schemes (resulting in a between-subjects analysis) and applied a paired t test for each coloring scheme pair in each search target type separately. All other differences were statistically insignificant except block vs. token scheme in the case of parameter target (paired t test, $t = 2.207, df = 11, p = .0495$).

In order to avoid the learning effect we also looked at the differences between different coloring schemes within the same presentation order (resulting again in between-subjects analysis). There were some statistically significant differences but these were contradictory. For example, in the assignment target case, the block scheme resulted in fastest searches if it was the first scheme presented to the participant ($F(2, 61) = 4.755, p = .0120$) but the same scheme resulted in the slowest searches if it was the second scheme ($F(2, 61) = 2.132, p = .1249$).

Correctness: Table 3 contains the mean correctness of answers broken down by the three search target types and the three coloring schemes. The means are also depicted in Figure 3. A two-way within-subjects ANOVA on correctness showed that there was a significant main effect of target type ($F(2, 30) = 9.920, p < .001$) but no main effect of coloring scheme nor interaction effects. A post-test with Bonferroni adjustment indicated that the main effect of target type was due to the higher correctness of assignment targets as compared with parameter and statement targets ($p < .05$).

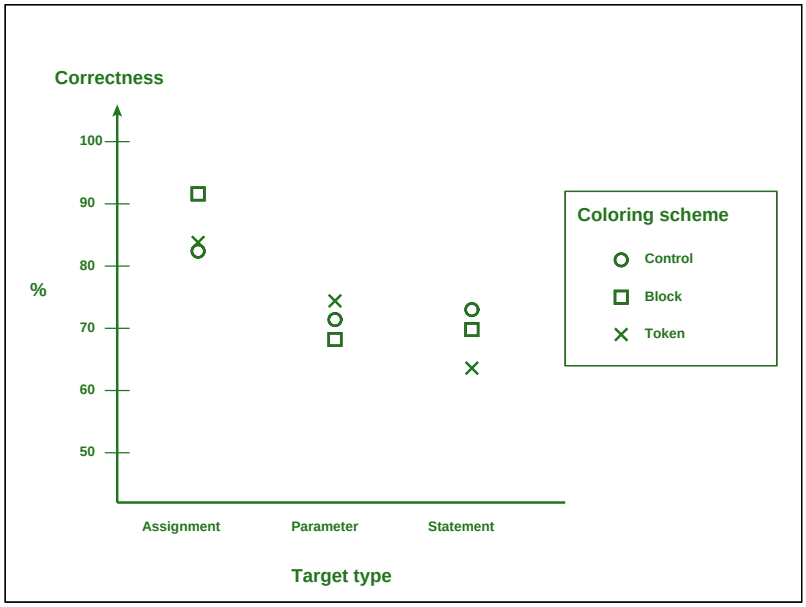

Fig. 3. Mean correctness in the three search target types for the three coloring schemes.

Table 3. Mean correctness percentages and standard deviations in different target types and coloring schemes.

	Target Type					
	Assignment		Parameter		Statement	
	Mean	S.D.	Mean	S.D.	Mean	S.D.
Control scheme	81.3	25.0	70.3	30.6	71.9	18.0
Block scheme	90.6	12.5	67.2	27.0	68.8	17.1
Token scheme	82.8	21.8	73.4	24.9	62.5	24.2

Table 4. Mean correctness percentages broken down by the presentation order of the different coloring schemes.

Order of presentation	Mean	S.D.
First coloring scheme	67.7	46.8
Second coloring scheme	75.9	42.8
Third coloring scheme	79.2	40.6

There was again a notable effect of the presentation order as seen in Table 4. To study this effect in more detail, a three-way between-subjects ANOVA using the presentation order as the third factor was performed. It showed a significant main effect of the target type ($F(2, 117) = 7.190, p = .001$), no main effect of the presentation order ($F(2, 117) = 1.647, p = .197$) but an al-

most significant two-way interaction of presentation order and coloring scheme ($F(4, 117) = 2.257, p = .067$).

4 Discussion

The task of the participants was to visually search for targets in program fragments shown as in an editor interface but with no editor functionality. The targets were local in the sense that they could be identified by looking at a single line or at most at a few consecutive lines. The phrasing of the tasks was carefully designed not to contain any of the strings belonging to the target of the search; thus the search was not purely visual but required a mental transformation from the task instructions to an attentional template [7]. As a result, the tasks resemble elementary operations in many programming activities that are accomplished with a program editor.

The three coloring schemes used in the experiment were not equally familiar to the participants. The control scheme—black text on white background—was familiar to all participants. Many had also used editors providing highlighting of lexical tokens even if not exactly the same scheme as the token scheme in the current experiment. The block scheme was new to all participants but it was conservatively designed with only two colors in addition to black: blue for comments (as in the token scheme) and red for method headings and declarations.

The search targets represented three types of patterns to be searched for: assignment targets were assignments of some specific form, parameter targets were method calls with a certain number of parameters; and statement targets considered control structures. These target types may not represent programmers' actual search targets well [11, 12] but they are examples of local search targets with different target sizes. For example, assignment targets corresponded typically to much shorter code sequences than parameter targets.

Differences among the three target types were significant both for search times ($p < .001$) and correctness ($p < .001$). Targets that corresponded to longer code sequences took also longer time to search for. Correctness did not, however, follow the same pattern. Even though the assignment targets that correspond to short code sequences required least time and provided best correctness, the parameter targets resulted in longest search times and the statement targets in poorest correctness. A possible explanation for the behavior of correctness is how well the target areas can be predicted. For assignment and parameter targets, the program code area, which needs to be studied to make sure that correct target has been found, is evident. In the case of statement targets, however, there are no clear boundaries which makes it easy to look at a too narrow code area and consequently accept a wrong piece of code. Thus statement targets are more error-prone than the other two target types.

None of the coloring schemes turned out to be superior to others. In fact each of them yielded the smallest search time and best correctness for some target type: block scheme was best for assignment targets, token scheme for parameter targets, and control scheme for statement targets. There may be many reasons

for this. For example, method declarations and calls used as parameter targets are vital in programming and participants may have developed long-term learned importance for the highlighting patterns in the familiar token scheme before participating in the experiment. As another example, indentation is an important cue for statement targets but the token scheme may lower indentation recognition efficiency which gives advantage to the other familiar coloring scheme—the control scheme. Finally, candidate solutions for the assignment targets can be detected by searching for a visually distinctive form, the character "=", which is hard in the cluttered token scheme. As the comments contained no special characters, search for the equal sign may be easiest in the block scheme.

The differences between coloring schemes were statistically insignificant except the shorter search times when using the token scheme as compared with the block scheme in searches for parameter targets ($p = .0495$). As the distribution of search targets in actual programming work is not known and as the differences between coloring schemes were small, there are no grounds to recommend any of the three schemes over any other.

A surprising finding was that the control scheme resulted in the same overall search performance as the other coloring schemes. This was unexpected because the other two coloring schemes use a special color for comments and thus provide an opportunity to skip parts of code during searches. The token scheme uses in special cases other colors within comments which may attract visual attention, but the block scheme uses a single color for comments that should thus be easy to skip during a search. However, the color used for comments was bright blue which may have caused a stimulus-driven pop-out effect [7]. A pop-out directs attention to the comment and results in automatic processing of the comment text, which takes time but is useless for the task at hand. The targets were hard and search had to be guided by goal-directed control. Thus interference caused by fault pop-outs is likely to happen. It seems that comments should not be highlighted in a bright color but rather "downlighted" with a color that does not attract visual attention—perhaps light gray if the background is white.

Eight of the 16 participants reported in their comments on color usage that they had found the colors useful during searches ("made searches easier", "three colors makes searches faster", ...), two participants wanted to see more colors, five gave a neutral comment ("diverse", "sufficient", ...), and one gave a negative comment ("not very good"). Thus the intuitive judgment of the participants was in favor of color usage but the results of the experiment do not support this intuition. It may be that colors make the screen aesthetically more pleasant and thus makes participants to experience the tasks easier than they actually are. If this is the case, the use of color can be justified by better work satisfaction.

It must be remembered that the current experiment dealt only with local searches. In other activities the effect of color may well be different. For example, in writing new code, the coloring of lexical tokens is highly practical: a misspelled keyword does not change its color to the color of keywords when completed, a missing quotation mark at the end of a string affects a pop-out effect as a large block of program code gets the color of strings, etc. These effects could, however,

be implemented using some more conservative coloring techniques: by coloring misspelled keywords (and, e.g., variable names) only, by coloring the first line of a string with missing quotation mark only, etc. Thus the coloring of lexical tokens could be used in the case of errors only, leaving a possibility to use other coding schemes for other purposes. Such coloring principles deserve more research.

There are several threats to the validity of this experiment. First, due to the overall similarity of the tasks and possible recall of earlier versions of tasks, there was a high learning effect making the comparison of coloring schemes problematic. To overcome this problem, we looked at individual tasks that were presented in the same order to all participants; only the ordering of coloring schemes was varied in the study. This technique allowed us to use a paired t test on the search times of individual tasks but lead to statistically weaker between-subjects testing. Another solution might be to have a much longer practicing period which, however, would lead to longer experimental sessions. That would be impractical because even now some participants reported that the experiment was too tiresome. It would also be possible to mix tasks of the three coloring schemes but this was not done because transitions from one coloring scheme to another cause unpredictable effects on viewing.

Second, the tasks were artificial in the sense that they do not necessarily correspond to actual search tasks in programming. The tasks were designed to require a transformation from a verbal description to a visual pattern so that they were not pure visual searches for a given visual pattern. On the other hand they did not require a full understanding of the program fragments so that the variance caused by a lengthy program comprehension process was avoided. Thus they required more processing than classical visual search tasks in cognitive psychology but less processing than programming related tasks in general. This way we tried to approach the basic search tasks in programming.

Third, the program code used in the experiment may not represent programs in general. To improve ecological validity, the code was not specially written for this experiment but downloaded from web. However, different programming styles are possible, and a final distribution version—like the one used in the experiment—may look very different from a program code that is under development. For example, when changes are made in code, many programmers first take a copy of the original, comment the copy out, and then change the original code. The copy is all the time visible and it looks just like normal code but there is no sense in looking at it during editing. Text that looks like program code has a long-term learned importance to programmers and can thus be expected to be hard to ignore [7] even if it is within a comment. In such a situation, it is very convenient to have a special color for comments to be able to make a quick distinction between real code and code saved as a comment. Thus a coloring scheme can improve the role-expressiveness of program text [13]: one color for actually working code, another for comments, and maybe another for declarations.

Fourth, the number of participants was small. It may be that statistically significant differences between the three coloring schemes were not found for this reason. On the other hand, statistically significant differences were found between

the three target types. Thus one can assume that even if the coloring schemes actually affected search times, the differences would be small when compared with differences between target types. As the distribution of search targets in actual programming work is not known, the conclusions would not change with better statistical significance of differences between coloring schemes.

Finally, the overall correctness of the tasks was low (74.2%). Thus the tasks were harder than expected which may result in uncontrolled effects during the searches.

5 Conclusion

We have presented the results of an experiment where different code highlighting techniques were used by intermediate programmers in visual search tasks for local patterns in Java program fragments. The targets were local in the sense that they could be identified by looking at a single line or at most at a few consecutive lines and they could be identified without an extensive comprehension of the program fragments. The search targets represented three types of patterns to be searched for: assignment targets were assignments of some specific form; parameter targets were method calls with a certain number of parameters; and statement targets considered control structures.

For code highlighting, three coloring schemes were used: highlighting of lexical tokens (token scheme), highlighting of larger syntactic constructs (block scheme), and no highlighting (control scheme). The first technique of giving different colors to different lexical token types is common in current program editors. Also the control condition is familiar to most programmers. The block scheme was new and had different colors for declarations, executable code, and comments.

Search performance differences among the three target types were significant but differences between the coloring schemes were small and not statistically significant. Each coloring scheme turned out to be best for some target type but the interaction was statistically only almost significant. As the distribution of search targets in actual programming work is not known, this study provides no grounds to recommend any of the three schemes over any other.

A surprising finding was that the control scheme, black text on white background, resulted in the same overall search performance as the other coloring schemes. The token and block schemes used in the experiment highlight comments with bright blue and this may cause pop-out effects that direct attention to comments and result in automatic processing of comment text. It seems that comments should not be highlighted in a bright color but rather "downlighted" with a color that does not attract visual attention.

The participants' overall comments on color usage were positive and they reported of perceived performance improvement. However, the results of the experiment do not support this intuition. Colors may be aesthetically pleasant and thus contribute to better work satisfaction but this does not necessarily mean improved performance. Likewise, highlighting principles of current program ed-

itors are based on their designers' intuition that can also be false. Ideas, which are based on intuition only, should be studied carefully before they are used as design guidelines.

The current experiment dealt only with local searches. In other activities the effect of color may well be different. Moreover, several coloring schemes could be used intermittently for better assistance in different tasks. The use of color in program editors deserves more research.

References

1. Green, T.R.G., Bellamy, R.K.E., Parker, J.M.: Parsing and gnisrap: A model of device use. In Olson, G.M., Sheppard, S., Soloway, E., eds.: Empirical Studies of Programmers: Second Workshop, Ablex Publishing Company (1987) 132–146
2. Cigas, J.F.: Dynamically displaying a Pascal program in color. In: Proc. of the 1990 ACM SIGSMALL/PC Symposium on Small Systems, ACM Press (1990) 68–71
3. Gilmore, D.J., Green, T.R.G.: Are 'programming plans' psychologically real – outside Pascal? In Bullinger, H.J., Shackel, B., eds.: Human Computer Interaction – INTERACT'87, Elsevier Science Publishers B.V. (1987) 497–503
4. Ehrlich, K., Soloway, E.: An empirical investigation of the tacit plan knowledge in programming. In Thomas, J.C., Schneider, M.L., eds.: Human Factors in Computer Systems. Norwood, NJ: Ablex Publishing Company (1984) 113–133
5. Keithen, K.B., Reitman, J.S., Rueter, H.H., Hirtle, S.C.: Knowledge organization and skill differences in computer programs. Cognitive Psychology **13** (1981) 307–325
6. Egeth, H.E., Yantis, S.: Visual attention: Control, representation, and time course. Annual Review of Psychology **48** (1997) 269–297
7. Desimone, R., Duncan, J.: Neural mechanisms of selective visual attention. Annual Review of Neuroscience **18** (1995) 193–222
8. Itti, L., Koch, C.: Computational modelling of visual attention. Nature Reviews Neuroscience **2** (2001) 194–203
9. WWW Site: Megamek. http://megamek.sourceforge.net/idx.php?pg=main/ (2006) (Accessed May 4th, 2006).
10. WWW Site: Vim the editor. http://www.vim.org/ (2006) (Accessed May 4th, 2006).
11. Singer, J., Lethbridge, T.: Studying work practices to assist tool design in software engineering. In: Sixth International Workshop on Program Comprehension IWPC'98, IEEE Press (1998) 173–179
12. Vans, A.M., von Mayrhauser, A., Somlo, G.: Program understanding behavior during corrective maintenance of large-scale software. International Journal of Human-Computer Studies **51** (1999) 31–70
13. Green, T.R.G., Petre, M.: Usability analysis of visual programming environments: A "cognitive dimensions" framework. Journal of Visual Languages and Computing **7** (1996) 131–174