Effects of syntax highlighting on comprehension of programming code

Abstract

This paper theorized that colour coding programming code (aka syntax highlighting) would make it *quicker* to *comprehend* what the code is doing for a programmer, something that might seem intuitive, but in order to avoid assumptions about its effect, and to firmly prove this effect, an experiment where participants had to complete a series of simple programming tasks, both colour coded (CC), and none-colour coded (NCC), where set up to answer this question. The effect of colour coding was also contrasted to the skill of the programmer. After conducting the experiment, the results show that there was a small difference in the time it took to complete colour coded tasks, being slightly smaller for syntax highlighted code, but this finding was not statistically significant. It was also found that the better the programmer, the less benefit (in time) he/she stood to gain from syntax highlighting.

Introduction

For this paper, syntax highlighting and colour coding is used somewhat interchangably when refering to colour coding of programming code. “NCC” and “CC” is used as abbrivations of “no colour coding” and “colour coding” respectively.

Syntax highlighting is a form of colour coding used by programmers to give keywords specific colours, the purpose being to add an extra element for identification of these keywords, in addition to the name itself. For example, the keyword “function”, denoting a function, might have the colour purple, while a constant, e.g. a number, might be colour coded with a yellow colour. This allows an additional way of separating between the two. The number of different colours used differ from syntax highlighter to syntax highlighter, aswell as the philosophy behind the creation of the syntax highlighter, be it opinions on whether high contrast is good or bad. An additional purpose of colour coding is to indicate mistakes. For example, if a keyword has not been spelled correctly it will also not be colour coded the expected way and thus gives additional visual information to the programmer that he/she has made a mistake.

There are a plethora of different syntax highlighters out there. Most, if not all, text editors and IDE’s (integrated development environment) come installed with some form of syntax highlighting. It seems wierd then; that the research supporting the various effects of such a

feature seems so elusive. It might be that the benefit of colour coding/syntax highlighting of programming code seems so intuitive to the common programmer, that it has been accepted as useful, and thus has become a staple in the many of the tools of programmers of the present day. Perhaps also a part of its allure is that it just looks better to some, and its benefit has become secondary. Given the lack of research on the topic, opinions and preference seems to have become the two main factors for choosing a syntax highlighter. Investigating the various effects of colour coding programming code, can help provide evidence of its effect and benefit, aswell as help shape the future creation of syntax highlighters.

Background

A (fairly old) quantitative analyses on the effects of colour on visual search and identification performance done by Christ (1975) indicated that “... color may be a very effective performance factor under some conditions, but that it can be detrimental under others.” In other words, research show that it is not a given that colour will aid a person, it could also be detrimental in certain circumstances.

A lot of the related research is old and not directly related. However, in more recent times a few papers on the subject matter have been written. Perhaps a reason for the surfacing of these studies now; is due to the increased affordabillity of eye-trackers. A tool some of the studies utilized.

Research on colour coding and visual search has been mostly not concerned itself with programming code. However, even though research regarding visual search and colour coding of programming code is limited, one such research was conducted by Hakala, Nykyri, and Sajaniemi (2006).They found, perhaps suprisingly ”... that the control scheme, black text on white background, resulted in the same overall search performance as the other coloring schemes.” This was despite the fact that the participants themselves reported a perceived notion of improved performance (Hakala, Nykyri, and Sajaniemi, 2006). Even though it might be intuitive that colour coding aids the programmer, it is not necessary the case. This research seem to weaken this sentiment.

For this research the interest lies in something more than just visual search, in arguably a more important quality of the syntax highlighter. The ability to aid understanding of the program. Much more recently, two studies Sarkar (2015) and Beelders and du Plessis (2016) has been conducted related to colour coding of programming code and understanding of the code. The goal of the former to investigate the impact of colour coding on comprehension of code and the goal of the latter to investigate whether learning materials for students, written most often in black and white, had a negative effect on learning outcome. Moreover research conducted by Dimitri (2015) investigated how debugging tasks and writing tasks was affected by syntax highlighting.

Most closely related to this research is the research done by Sarkar (2015) which did indeed find a significant difference between the time to comprehend colour coded and non-colour coded, a weakness of this research however was the limited participant sample size, consisting of only 10 participants.

Research question

As seen, research on effects of syntax highlighting exists, but is sparse. Most of the research is just loosely related, but not directly tied to colour coding of programming code. Although colour coding has shown to help aid visual search under some conditions (Christ, 1975), there is little research that shows whether this benefit can extend to something like aiding *comprehension* of what you are reading. Furthermore, reading of programming code differ from reading of traditional text with a lot more skips as well as back and forth eye movement when reading, it is not strictly a linear reading process.

In order to delve into the nitty gritty of how syntax highlighting affects performance and what and how the different properties of a syntax highlighter affects performance, there is a need to firmly prove the usefulness of syntax highlighting in itself. Moreover, the limited available literature on the subject also warrants proper research into the subject.

In order to increase understanding of the effects of colour coding on programming code questions like: What is the effect of syntax highlighters. Is it faster to understand code that uses CC than code that uses NCC? Is this difference diminished with the increased skill of the programmer? is highly relevant.

In order to investigate the effect of syntax highlighting on programming code the following hypothesis was formulated:
Colour coding of programming code reduces the time it takes a programmer to comprehend the meaning of a program/piece of code.

To answer the secondary aim of the study the following hypothesis was formulated:
With increased skill of the programmer this benefit in time is reduced.

Defining effect as time
The effect of colour coding, negatively or positively related, can be measured in many ways. In order to have some tangible metric that could be used to show the potential usefulness of syntax highlighters, not just preference, *time* to complete tasks was measured. By comparing the time it would take to complete tasks with both NCC and CC, a comparison could be made to see if one was superior to the other in terms of the time it took to comprehend what the program was doing.

Defining skill
The research also posed the question of whether there were conditions under which the hypotizised effect of colour coding might be diminished. The condition chosen was skill. With skill more than just experience is ment. Skill is defined here as the sum of all elements that would affect how well you did performing the test. More accurately, skill is defined as how fast a person completed the test, the total time, compared to the rest of the participants. By measuring the time it took for the participant to complete the entire experiment, the sum of all tasks, we had a measurable, unbiased measure of skill, one that encompassed more than just experience, something that other research like Sarkar (2015) had looked into. This value also captured tiredness, experience, nervousness, familiarity with the specific language used, how long ago since they programmed or read code, and any other variable that would affect the performance at the time of taking the test. Measuring this variable, the total time of completion for the entire test, offered an objective, unbiased way of measuring how good/skilled the participant was at that moment in time. A self evaluation approach, akin to what Sarkar (2015) did for measuring experience, could not feasibly capture all of these variables, and would also be prone to bias and inaccuracy.

Experiment design

Given the experimental design; tasks were made to be small. They could fit on the screen without scrolling, and consisted of figuring out the value of a given variable. There were in total 6 tasks. The tasks appeared in pairs, that were made equally difficulty, but had different output and meaning, requiring the participant to understand the program in either case. This was achieved by changing some values and by reversing operations, additions to subtractions etc, similar to Sarkar (2015).

The tasks were made even easier than what was expected of a novice, this was because McCracken et al. (2001) had discovered that many students did not know how to program at the conclusion of their introductory courses, and students that had just completed their introductory courses would be a part of the samples for this study. At the same time it was important that the tasks were hard enough that they were not too easy for experts or intermediates, as well as hard enough that the functionality of the program were not obvious at first glance, making the task more than just a visual search.

Names of variables were kept consistent among tasks, and also anonymous, not giving away meaning by themselves. The background colour was kept the same between the NCC and CC. The code snippets for CC consisted of 5 different colours (including the colour which was the one used for NCC). The syntax highlighter used was the One Dark Syntax theme (2015). Javascript was choosen as the language for the test, since most of the participants were guessed to be familiar with this language.

A pilot test was conducted during which some problems were identified. Due to this, some of the tasks were altered, some small mistakes that might have added confusion was changed,

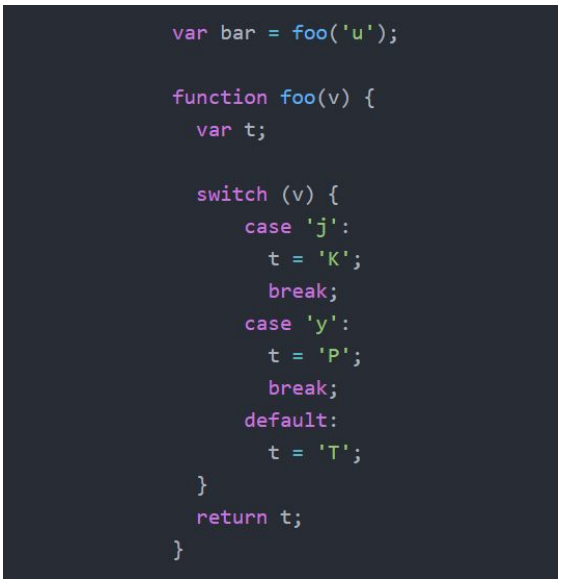
and the task with two loops inside eachother was changed to just one loop, to make it easier for the participants to comprehend program functionality, while still maintaining enough difficulty to not make the solution obvious, and to better accomodate the different skill levels of the test participants.

The experiment

The experiment had a total of 20 participants. All participants were students at university level. Some bachelor, some master level students. The participants had varying levels of skill. An incentive in the form of chocolate was offered in return for the participants time.

The experiment was conducted in person, in a quite environment with no distractions, using the same equipment for each participant (a laptop), creating the same consistent viewing condition. Each participant was given the same instructions. They were informed that the test was timed, but that they should focus on getting the correct answer. Those who were unfamiliar with Javascript as a language was explained the small differences to other languages. The tasks were designed to be as independent of the language as possible. The test consisted of a series of small code reading tasks, difficult enough to require deciphering, in order to measure comprehension. They were presented in sequence, with a chance for pausing inbetween. There were a total of 6 tasks, and each participant solved all 6 of these tasks by answering the posed question: “What is the value of the variable ‘bar’”. The time to complete tasks were recorded in milliseconds. The order of the colour coded vs none-colour coded tasks were randomized to account for potential learning effects.

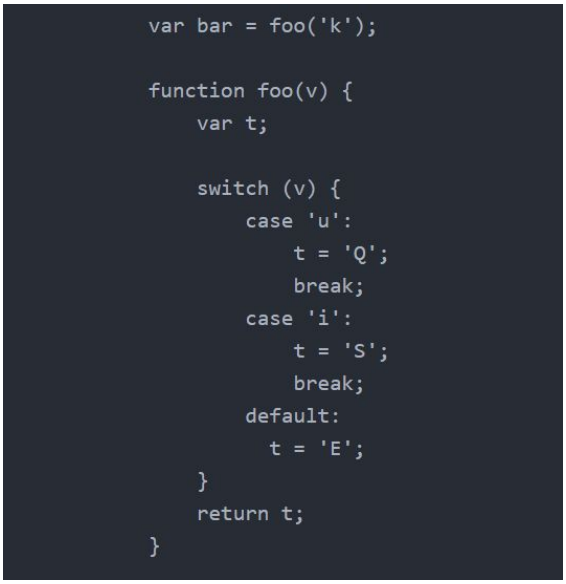
Below is two of the tasks from the test, one with, and one without syntax highlighting.



```
var bar = foo('u');

function foo(v) {
  var t;

  switch (v) {
      case 'j':
        t = 'K';
        break;
      case 'y':
        t = 'P';
        break;
      default:
        t = 'T';
  }
  return t;
}
```



```
var bar = foo('k');

function foo(v) {
    var t;

    switch (v) {
        case 'u':
            t = 'Q';
            break;
        case 'i':
            t = 'S';
            break;
        default:
          t = 'E';
    }
    return t;
}
```

Results

Each of the 6 tasks took on average of 40 seconds to complete +/- (1-3) seconds.
A statistical analysis (Paired Samples t Test) of each pair separately, revealed that for the first pair neither colour coding nor none-colour coding provided a statistical significance average difference in the time it took to complete the tasks ($p < 0,883$). The difference between the means were 1,4 seconds. The syntax highlighted task being the quickest to complete.

Likewise did the second pair not provide a significant average difference ($p < 0,800$) in the time it took to complete each task. Again the difference was small. The difference between the means were 1,8 seconds. The syntax highlighted task being the quickest to complete.

The last pair did also not provide any significant average difference between the time to complete the tasks. The difference between the means were 3,4 seconds. The syntax highlighted task being the quickest to complete ($p < 0,571$).

Taking every task pair into consideration and performing the same test for the whole experiment,, the result is that colour coded tasks took in total 117,99 seconds to complete while no-color coded tasks took 124,52 seconds. This was not seen as significant either ($p < 0,730$).

Additionally, this study asked itself the question of whether this potential benefit from colour coded programming code would extend to skilled programmers as much as novice programmers. Performing a Pearson correlation coefficients test between the time to complete the entire test (skill) and the difference between the time to complete colour and non-colour coded tasks (benefit of colour coding) revealed a positive moderate correlation between these two variables. This finding was significant at the 0.05 level ($p < 0,021$).

Discussion

A small difference of colour coded tasks being 2-3 seconds on average quicker to complete than non-color coded ones was found performing a statistical analysis. However, this did not make for a significant difference where it was possible to conclude that one colour scheme was significantly faster than than the other to complete the task with. This seems perhaps to go against intuition. Reasons for this could be the size of the task. It could be that this small difference would be significant if the task/program increased in size, and that simply because of the simplicity of what was needed to be comprehended it did not offer enough of a challenge that the colour coding gave a big enough advantage, only a small one.

In order to perhaps get a clearer picture of whether colour coding does indeed reduce the time it takes to comprehend a program, increasing the size and difficulty of the program might help. Although this would require more experienced programmers.

Results showed that novice programmers benefitted more from the syntax highlighted code than those who were intermediates or experts. The less time the participant used on the test, the less difference in time there were between the NCC and CC tasks for that participant. This was the proposed prediction and hypothesis at the start of the study. This seemes to confirm the intuition that colour coding helps more for novice programmers. The crude explanation for this could be that someone who can be considered a good programmer would *need* the colour coding less in order to be able to understand the meaning of the program, but more research is needed to know why this is. The scope of this study only looked at whether such a difference existed.

It should also be noted that these tasks was not an accurate representation of common programming tasks. They were made for this test specifically, but are not the same tasks that a programmer would necessarily face when comprehending code in their everyday life. If they were encountered, they would be the part of a much bigger program. These code snippets on their own does not carry much value.

Another possible concern is that tasks were not equally hard. There is little reason to believe so, seeing as the tasks are essentially the same, just with different values and reversed operations. There is, however, always the chance that someone will think subtraction is harder then addition for example, even if the numbers are small. Also, the mean difference was roughly the same for all pairs adding to the reason to believe the tasks was equally difficult.

Conclusion

In the present study, colour coded tasks were found to be slightly quicker to complete than none-colour coded tasks, but not enough to be considered significant. It took only a few, 1.5 - 3, seconds faster to complete syntax highlighted tasks. However, a moderate correlation, significant at the 0.05 level, were found between the skill of the programmer and the benefit of colour coding. Seeing as this is the case, with bigger, more difficult tasks, or with more novice programmers, a significant difference to complete tasks with the different colour schemes might be found with this increased difficulty of the tasks.

A recommendation then becomes that future research includes more difficult tasks or longer code snippets, to see the effect that this will make. Even though this study improved upon other studies like Sarkar (2015) by having more subjects in the study, this is something that can be improved upon further in future studies. Hopefully, with the increased affordability of eye-trackers, we might see more research of this nature in the future with the new ways of studying both text and visual search that eye-trackers offer.

References

Beelders, T. E. and du Plessis, J-P. L. (2016) Syntax highlighting as an influencing factor when reading and comprehending source code, *Journal of Eye Movement Research*, 9(1), pp. 1-11. doi: http://dx.doi.org/10.16910/jemr.9.1.1

Christ R. E. (1975) Review and Analysis of Color Coding Research for Visual Displays, *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 17(6), pp. 542-570. doi: https://doi.org/10.1177/001872087501700602

Dimitri, G.M. (2015) PPIG 2015: The impact of syntax highlighting in Sonic Pi. *Proceedings of the 26th Annual Conference of the Psychology of Programming Interest Group (PPIG 2015)*. Bournemouth, July 15-17, 2015. Bournemouth: PPIG, pp. 59 – 68.

Hakala, T., Nykyri, P., and Sajaniemi, J. (2006) An Experiment on the Effects of Program Code Highlighting on Visual Search for Local Patterns, *18th Workshop of the Psychology of Programming Interest Group*. Sussex, September 7-8, 2006. Sussex: PPIG, pp. 38–52.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Ben-David Kolikant, Y., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001) A multi-national, multi-institutional study of assessment of programming skills of first-year CS students, *Working group reports from ITiCSE on Innovation and technology in computer science education (ITiCSE-WGR '01)*. Canterbury, June 25-27, 2001. New York: ACM, pp. 125-180.

One Dark Syntax theme (2015) Available at: https://atom.io/themes/one-dark-syntax (accessed: 19. march 2018).

Sarkar, A. (2015) The impact of syntax colouring on program comprehension, *Proceedings of the 26th Annual Conference of the Psychology of Programming Interest Group (PPIG 2015)*. Bournemouth, July 15-17, 2015. Bournemouth: PPIG, pp. 49-58.