Effects of colour-coding on programming comprehension

Abstract

Source code is often colour-coded with the promise of helping the programmer in both reading and writing code. The current work set out to explore the benefits of such colour-coding, both in the form of syntax as well as semantic highlighting. An experiment was set up in order to investigate the effects. Results showed that semantic highlighting significantly reduced the time to comprehend source code, while contrary to previous work, syntax did not. The results also showed that this effect diminished with the skill of the programmer with the exception of semantic over syntactic.

Keywords: syntax highlighting, semantic highlighting, colour-coding, programming, human factors, usability

1. Introduction

What is syntax highlighting

Syntax highlighting is the colour-coding of the syntax of progamming source code (sometimes also called “prettified code”). It is a standard feature of text editors or integrated development environments (IDEs) ment for writing programming code. Many of these editors come with pre-installed syntax highlighting, but most modern day editors also allows the user to install their own syntax highlighter or modify the current one. Syntax highlighting is the most common form of colour-coding programming code.

```
shuffleCards = function(array) {
    var m = array.length, t, i;

    while (m) {
        i = Math.floor(Math.random() * m--);
        t = array[m];
        array[m] = array[i];
        array[i] = t;
    }
    return array;
};
```

Fig 1. Code using a syntax highlighter used by Github[1]

Specifically, syntax highlighters colour certain keywords (tokens) for easier recognition. The colour does not, in any way, alter the meaning of the words that it colours, it serves only as an additional way to identify keywords by. Special keywords, e.g. “while”, is assigned one colour, names of variables another, numbers a third colour and so on and so forth.

Colour-coding the programming code has some additional benefits. If the programmer makes a mistake when writing, the colour-coding offers the text editor the ability to “highlight” this mistake. For example, if the programmer forgets to close a string, the text following the string will have the same colour as the string.

[1] Github - https://github.com/

Fig 2. Right: the text being coloured green shows the programmer that the he/she has forgotten to close the string

Different variations of current syntax highlighters

Syntax highlighters come in many variations. There are no standard concerning how many colours that is used. One highlighter can create it's colour palette using 8 colours, another 16. The colours and the contrast between them (and the background) also varies. Some highlighters utilize high contrast (both hue and/or brightness) between the coloured tokens. Such an example would be Solarized[2] which uses low brightness, but high hue contrast between the colours of its palette. Other highlighters utilize low contrast between the colours of its palette, e.g. monochromatic syntax highlighters.

Alternative highlighters

Although the form of colour-coding outlined above is by far the most common approach to aiding the programmer in understanding the code, alternatives have been developed. For example, it is possible to colour words based on the meaning they have (without changing the meaning), or based on the context they occur in. In more recent times, Nolden (2009) proposed semantic highlighting. Semantic highlighting gives local variables their own unique colour, and in this way extends syntax highlighting. The purpose is to utilizes colour in order to highlight some meaning of the coloured token (word), not just what type of token it is. When a local variable is given an unique colour; the programmer can more easily see where this local variable occurs in the code and more easily get an overview of the variables usage and purpose - atleast, thats the idea.

A consequence of giving local variables their own unique colour is that more colours than what is easily separable is needed. For example, we can easily identify yellow and blue as separate colours, but two different shades of blue are harder to separate. This is especially relevant when we are *talking* about these colours, but less so when we are visually seeing the colours. They might not be as obviously different at first glance due to lower hue contrast, but visually we can still see the difference.

[2] Ethan Schoonover - Solarized - https://ethanschoonover.com/solarized/ (retrieved 24.10.18)

Programming tasks in relation to colour-coding

The interaction of the programmer and the source code happens in a few ways. A programmer does more than simply write code. He/she also reads their own code, others code, as well as debug said code. Reading can include visually scanning for something specific or reading line for line. Debugging includes identifying errors. What is important is the focus on understanding the code (code denotes an algorithm), which is a lot more complicated than reading and understanding prose and linear text. Syntax highlighters play a role in all of these tasks as a possible enhancer of performance and are thus relevant to study the effects of.

1.1 Background

The effects of syntax highlighting has gained more attention by researchers in recent years, though the earliest research on the topic was done back in the 80's and 90's (Hannebauer, Hesenius and Gruhn, 2018). Despite this, and despite the recent increase in research on the topic in the last few years, there still seem to be a lack of research available.

Among the existing work on syntax highlighters and their various effects and benefits, Beelders and du Plessis (2016) studied how syntax highlighting influences source code reading and scanning behaviour. Based on the result, they drew the conclusion that in both scanning the code, and reading the code, the *lack* of syntax highlighting did not adversely affect the behaviour. Based on this, questions concerning the usefulness of syntax highlighting can be raised.

Beck et al. (2014) used visual augmentation (as they describe it) to enhance the ease of comprehension of regular expressions, a programming technique, which, due to its complexity and compactness, can be difficult to read and understand. The logic was that using syntax highlighting in this source code context could make it easier to read this form of notation. They did not prove statistical significantly improved results using this approach, but gathered expert feedback; which indicated that the readability of regular expression was increased.

Sarkar (2015) investigated whether the presence of syntax highlighting affected the time to complete tasks in small python programs. They performed a statistical analysis on the data, and found that syntax highlighting significantly reduced the time to complete the tasks for the

participants. They also found that the variable of experience, reduced this benefit. However, only 10 participants partook in the study, which might put the validity to question.

Dimitri (2015) performed an empirical study where they investigated the role of syntax highlighting in Sonic Pi. The participants both performed writing and debugging tasks and found that syntax highlighting significantly improved the task completion time for both tasks. Two things are valuable to note. Firstly, the writing and debugging of Sonic Pi code is a much simpler task than in most programming languages used for commerical products and services. Dimitri (2015) notes that Sonic Pi is primarily ment for children. The second thing to note with this study was that, once again, the participant sample size was relatively small; with only 10 participants.

Both Sarkar (2015) and Dimitri (2015) highlights the possiblity that the effects they found might increase with the increase in code length, and both recommend this as future research. This can also be a reason as to why not all studies (Beelders and du Plessis, 2016) has found the same effect.

Asenov, Hilliges and Müller (2016) investigated the effect syntax highlighting had on the comprehension of code in reading tasks. They compared a mainstream syntax highlighter to two visually enhanced versions, by asking questions about code structure. The visual enhancement was done by both utilizing colour as well as other ways to represent the code (e.g. visually outlining blocks instead of using curly brackets). The results found that visually enhancing the code did significantly decrease the time to answer a wide range of questions about code structure , and that contrary to an often held belief, that more colour creates “visual overload”, this was not the case. The results suggests that there is a lot of potential benefit in representing code differently than the standard way that editors and tools choose to do today.

The related, previous research, reveals two areas for exploration. Firstly, although most of the research focused on syntax highlighting specifically as the way of colour-coding programming code, this work is still lacking and warrants better investigation, not least because of the low number of participants and the somewhat ambigious results that different studies found (Beelders and du Plessis, 2016) versus (Sarkar, 2015) and (Dimitri, 2015).

Secondly, an area where we can clearly see the need for research, is related to alternative colour-coding other than syntax highlighting, e.g. semantic highlighting. To the author’s best

knowledge, no academic research exists on semantic highlighters effectiveness for programming. As a fairly new way of highlighting source code, semantic highlighters seem to have escaped scrutiny.

Given the lack of research on the topic, and since semantic highlighters potentially offers a new and improved way to view source code, research questions were formulated to give answer to the effectiveness of both syntactic and semantic highlighting.

1.2 Research questions

Although the interaction with source code happens in different ways, and for different reasons, the most important task is still *comprehending* the code. The colours of certain keywords can help to more quickly identify keywords, but this is of little use if there is no improvement in the time it takes to understand what the code actually does. To help answer the ambigious results on the effectiveness of syntax highlighters when reading code, as well as investigating the effectiveness of semantic highlighting, two research questions were formulated.

RQ1: Does syntax highlighters decrease the *time* to answer questions about *code output*?
RQ2: Does semantic highlighters decrease the *time* to answer questions about *code output*?

Furthermore, the relationship between the effect of highlighters and the experience and skill of the programmer is also relevant to the usefulness of such highlighters. Does the usefulness dimish with increased programming abilites? Previous research has found that experience is negatively related to the benefit in time gained from the use of syntax highlighting (Sarkar, 2015) and (Dimitri, 2015), but in both experiments self-reporting was used as the measurement of experience, which is open for erroneous assesment. The question of how valuable experience really is, can also be raised, after all, what one person can accomplish in 5 years, another can do in 2 years. Therefore we propose that the *skill* of the programmer is more valuable to know, rather than *experience*. That is why, for the present work, we are more interested in what we call *skill*, which is operationalized here as the total time to complete all the tasks. This way, if the programmer is tired, or has not used a particular language in some time, or for any other reason underperforms, the result of a test, when using time as measurement, will reflect this, even if the participant would score high on experience in a self-assessment approach. To confirm the

results of previous studies with this new way of looking at the performance of the programmer, a last research question was put forward.

RQ 3: Is the potential benefit of RQ1 and RQ2 diminished by the skill of the programmer?

2. Method

An experiment was set up to answer the research questions. It consisted of a series of tasks presented in sequence via an online tool programmed for this purpose. Utilizing a programmed solution like this, the time to complete task could be recorded down to the millisecond accuracy. The test tasks was done in Javascript, a language most of the participants were familiar with. To avoid any potential confusion however, they were informed of any differences to other common languages, that was native to JS. The differences was small, however, and the tasks were deliberately designed so that the language used should matter as little as possible. The order of the tasks was the same for each participant, but what highlighting type (semantic, syntactic and no-colour) that was selected for each task was randomized to account for order bias and task difficulty bias, with each type appearing two times each. The same task, for different participants, could therefore either be highlighted using syntax highlighting, semantical highlighting or have no highlighting at all. The colouring of keywords (e.g. the keyword "function" in fig. 3) between syntactic and semantic highlighting was kept the same, making sure only the variable colouring (the difference between semantic and syntactic highlighters) was different between them. The tasks asked the participants to identify the output of Javascript programs. This required the participant to understand what the program did, in other words, the meaning of the code, not just recognize code structure. The participants were informed that the test was timed.

The task variables where named using Metasyntactic names[3], so that the names did not give away meaning and helped making smaller task a little harder to comprehend, requiring more mental work. Indentation, font and bracket placements all stayed consistent throughout the test as well.

[3] https://en.wikipedia.org/wiki/Metasyntactic_variable

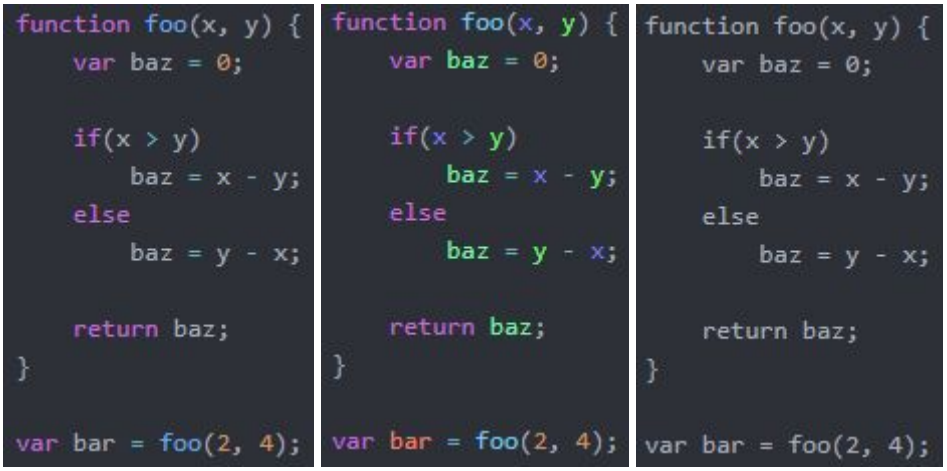

Fig 3: An experiment code snippet, from left to right, highlighted syntactically, semantically and without highlighting.

Participants:
The experiments participants was mostly students from all three bachelor years at NTNU. Additonally a few of the participants worked as programmers and had more experience than student level. The participants were all male and between 20 - 30 years.

3. Results

In order to answer RQ1 and RQ2 the sum time of all the tasks for each colour scheme, was first summarized and then a testing of the means for statistical significance was performed. First we compared the tasks with no colour-coding to the tasks highlighted syntactically. The mean time to complete non-coloured tasks was 97 seconds, while the time to complete the syntactic tasks were 122 seconds. Performing a paired samples t-test showed that this difference in mean time was statistically significant at $p < 0{,}003$, meaning that the tasks with no-colour was significantly faster than those with syntactic highlighters. Syntactic highlighters did *not* decrease the time to answer questions about code output.

Secondly, we compared again the tasks with no colour, this time to the ones semantically highlighted. The mean time to complete all these tasks was 97 seconds and 88 seconds respectively. Again performing a paired sample t-test these results were revealed as not

statistically significant (p < ,167), despite semantic being the faster tasks to complete by 9 seconds.

We already compared the time for no-colour and semantic highlighting, but to fully answer RQ2 we also did a test for significance between the time to complete syntax highlighted tasks and semantically highlighted tasks to find out if semantic truly was an "enhanced" version of syntactic.

The mean time to complete *all* the syntactic tasks was 122 seconds, while the mean time to complete the semantic tasks were 88 seconds. The paired samples t-test revelead a statistical significance at the p < 0,001 level, meaning that semantic was significantly faster than syntactic with a less than 1% chance for error.

In order to answer the last RQ we needed to first calculate the difference in mean time between the three colour schemes and then compare these new variables to the total test time (defined here as skill) to see if there were any correlation, negative or positive, between the benefit one gained from highlighting code and the skill of the programmer.

Performing a Bivariate Correlation test (Pearson) on the time difference between no-colour and syntax tasks and the total time to complete the entire test, which we defined here as skill, revealed that they were positively correlated with a Pearson's correlation coefficient of 0,672, considered to be a strong correlation, this was also considered significant at the 0,01 level (p < 0,001). This ment that when the benefit gained from no-colour increased so did the total test time, meaning that the benefit was strongest for those less skilled/the slowest performers.

A likewise test of Pearson correlation coefficient for the time difference between no-colour tasks and semantic tasks confirmed a coefficient of 0,803, again significant at the 0,01 level, revealing that the time benefit gained from semantic tasks previously found did diminish with skill.

Lastly a third test of the correlation between syntax and semantic was performed. This time the results were revealed as *not* significant at more than 99 percent chance of randomness with a correlation of 0,001. The time benefit gained from semantic highlighting did not diminish with skill.

4. Discussion

Results

The results confirm our hypothesis that semantic highlighting would decrease the time to comphrened code in the form of understanding code output. Semantic highlighting was in our test superior in this sense to both syntactic and no colour-coding. No previous research to our knowledge exists and therefore we cannot cross-reference this result. However, it does lend some credibility to the usefulness of semantic highlighting and indicates that it is worth studying this rarely used form of colour-coding further.

On the other hand, our experiment did not confirm RQ1. Syntax highlighting was inferior to both no-colour and semantic highlighting. This is a somewhat confusing result and goes against the previous research mentioned, since they found syntactic to be faster to read than no colour and the work that didn't by Beelders and du Plessis (2016), found no difference. Considering this it might be more reasonable to assume that our experiment had confounding variables or undisclosed biases, rather than conclude that the present work disproves previously drawn conclusions. One thing that can also be mentioned is that, considering the amount of randomization used to account for biases, it might have been necessary to increase the number of participants (or number of tasks or task length) to get clearer results. This does not fully explain why no-colour would be *significantly* faster than syntax highlighting, however.

When it comes to skill on the other hand we did seem to confirm our last research question, but not entirely since the benefit gained *did* diminish with skill, but not for semantic over syntactic. Our new way of looking at skill is then mostly in accordance with previous evidence and presumably shows us that highlighting code might not be so important when the programmer is past the novice stage. However, if there is truth to Beelders and du Plessis's (2016) work then this would indicate that syntax highlighting might actually not be that important to novices either, although that does not mean entirely useless.

5. Conclusion

Summary

The present work studied different forms of colour-coding programming code to aid the programmer. We found mixed results, where suprisingly, no colour-coding proved better than syntax highlighting, although semantic highlighting proved most benefical overall in reducing the time to comprehend the output of a piece of code. Furthermore, we also confirmed previous studies that more experience reduced the benefit gained, for two of the three cases.

Future work

Hannebauer, Hesenius and Gruhn (2018) highlights that, even if a syntax highlighter is of no benefit, this does not mean that it would be completely "harmless" to ignore this fact. By using a particular form of syntax highlighting that has no positive effect, we deprive ourself of the chance to use a colour scheme that actually does have a positive effect. The current colouring scheme (colouring tokens for easier recognition), is not the only possible scheme, and there might be better alternatives for colour-coding source code, which would better aid the programmer. This work studied one such alternative called semantic highlighting. Given that such alternatives to the current standard syntax highlighting approach was found to be beneficial both by the present work and by Asenov, Hilliges and Müller (2016), there is indication that efforts into studying other approaches to representing code, could be fruitful, and should be studied more systematically in future work.

6. References

Asenov, D., Hilliges, O., and Müller, P. (2016) The Effect of Richer Visualizations on Code Comprehension. *In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. San Jose, California, USA, May 07-12, 2016. New York: ACM, pp. 5040-5045.

Beelders, T. and du Plessis, J-P. (2016) The Influence of Syntax Highlighting on Scanning and Reading Behaviour for Source Code, *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT '16)*. Johannesburg, South Africa, September 26-28, 2016. New York: ACM, pp. 5:1 - 5:10.

Beck, F., Gulan, S., Biegel, B., Baltes, S. and Weiskopf, D. (2014) RegViz: visual debugging of regular expressions, *ICSE Companion 2014: Companion Proceedings of the 36th International Conference on Software Engineering*. Hyderabad, India, May 31 - June 07, 2014. New York: ACM, pp. 504-507.

Dimitri, G. M. (2015) The impact of syntax highlighting in sonic pi. *PPIG 2015 - 26th Annual Workshop*, Bournemouth, UK, July 15-17, 2015. Bournemouth: PPIG, pp. 59-70.

Hannebauer, C,. Hesenius, M. and Gruhn, V. (2018) Does syntax highlighting help programming novices?, *Empirical Software Engineering,* 23(5), pp. 2795 - 2828. doi: https://doi.org/10.1007/s10664-017-9579-0

Nolden, D. (2009) *C++ IDE Evolution: From Syntax Highlighting to Semantic Highlighting* Available at: https://zwabel.wordpress.com/2009/01/08/c-ide-evolution-from-syntax-highlighting-to-semantic-highlighting/ (accessed 15 March 2019).

Sarkar, A. (2015) The impact of syntax colouring on program comprehension. *PPIG 2015 - 26th Annual Workshop*, Bournemouth, UK, July 15-17, 2015. Bournemouth: PPIG, pp. 49–58.