Magdalena Andrzejewska

Kolorowanie kodu źródłowego programu a proces jego analizy syntaktycznej – badania eye-trackingowe

Edukacja - Technika - Informatyka nr 3(17), 204-209

2016

© Wydawnictwo UR 2016 „Edukacja – Technika – Informatyka" nr 3/17/2016
ISSN 2080-9069 www.eti.rzeszow.pl
ISSN 2450-9221 online DOI: 10.15584/eti.2016.3.31

MAGDALENA ANDRZEJEWSKA

Kolorowanie kodu źródłowego programu a proces jego analizy syntaktycznej – badania eye-trackingowe

The influence of syntax highlighting on reading program source code – eye-tracking research

Doktor, Uniwersytet Pedagogiczny im. Komisji Edukacji Narodowej w Krakowie, Instytut Informatyki, Polska

Streszczenie

W artykule przedstawiono wyniki eksperymentu, w którym badano wpływ kolorowania kodu programu na proces jego analizy składniowej. Uczestnikami eksperymentu było 13 studentów kierunku informatyka. Zastosowano technikę okulografii w celu śledzenia procesu wyszukiwania błędów syntaktycznych. Zaobserwowano, że podświetlanie składni nie wpłynęło na efektywność wykonania zadania. Nie stwierdzono również istotnych różnic w wartościach parametrów eye--trackingowych w przypadku analizy kodu czarno-białego i kolorowego.

Słowa kluczowe: uczenie się programowania, kolorowanie składni kodu programu, okulografia, analiza kodu programu.

Abstract

This article presents an empirical study investigating the role of syntax highlighting program source code. Eye-trackig data were collected from 13 computer science students, who were asked to search syntax errors in coloured and black-and-white code, while their screens were recorded. It was observed that syntax highlighting has not significant effect on the task execution. The eye--tracking parameters for black-and-white code and for coloured code differed, but not significantly.

Key words: learning programming, syntax highlighting, eye-tracking, source code review.

Wstęp

Programowanie jest złożoną umiejętnością uwarunkowaną nie tylko znajomością składni wybranego języka oraz opanowaniem takich czynności, jak edycja kodu, jego kompilowanie i uruchamianie. To również kompetencje związane z mechanizmami rozwiązywania problemów oraz konstrukcją i reprezentacją algorytmów. Nieopanowanie ostatnich z wymienionych umiejętności jest powszechnie uznawane za podstawową przyczynę trudności towarzyszących nauce programowania [Gomes, Mendes 2007]. Z poglądem tym polemizują R. Lister i wsp., którzy w swoich

badaniach ustalili, że studenci często wykazują brak dostatecznej wiedzy i umiejętności odnoszących się do procesu czytania kodu. Według badaczy to właśnie tę sprawność należy kształtować w pierwszej kolejności jako poprzedzającą umiejętności pisania własnego kodu i rozwiązywania problemów [Lister i in. 2004: 143].

Czy formatowanie kodu wpływa na efektywność jego analizowania (czytania) oraz rozumienia? Pierwsze badania dotyczące zagadnienia czytelności kodu programu prowadzone były w latach 80. ubiegłego stulecia. Ustalono wtedy, że istotne znaczenie ma tutaj stosowanie wcięć w obrębie linii programu [Miara i in. 1983: 867], wyróżnianie, np. kolorowanie, wybranych części [Rambally 1986: 176] czy odpowiedni układ graficzny poszczególnych elementów programu (typografia kodu) [Baecker, Marcus 1986: 57; Oman, Cook 1990: 519].

Wraz z rozwojem środowisk graficznych problematyka ta była przedmiotem zainteresowania w kolejnych badaniach, w których analizowano takie elementy formatowania tekstu programu, jak np. układ (*layout*) i wcięcia, krój i rodzaj czcionki, wygładzanie krawędzi czcionki (*anti-aliasing*), podświetlanie składni (*syntax highlighting*)i podświetlanie elementów zgodnie z regułami semantyki (*semantic highlighting*) [zob. Jacques, Kristensson 2015: 27–30]. Ustalono np., że odpowiednia struktura i składnia kodu źródłowego pozwala na szybsze jego zrozumienie niezależnie od zastosowanego stylu nazewnictwa identyfikatorów [Binkley i in. 2013: 219–276].

Od ponad dekady w badaniach nad mechanizmami analizy kodu wykorzystuje się metody okulografii [zob. Binkley i in. 2013: 264–265], a ostatnio również przeprowadza się eksperymenty z zastosowaniem funkcjonalnego rezonansu magnetycznego (fMRI) [Siegmund i in. 2014: 378–389].

Rezultaty badań okulograficznych pokazują, że proces czytania i rozumienia kodu źródłowego programu różni się zasadniczo od czytania i rozumienia języka naturalnego. Ale różnice te zanikają, jeśli tekst programu staje się podobny do tekstu języka naturalnego [Binkley i in. 2013: 220]. Nie ma natomiast jednoznacznych doniesień dotyczących wartości podstawowych miar okulograficznych (parametrów ruchów oka) odnoszących się do czytania kodów programów. Przykładem może być tutaj średnia długość czasu fiksacji, która w przypadku niektórych badań była znacznie wyższa (ok. 350 ms), a w innych zbliżona (100–200 ms) do wartości charakterystycznej dla procesu przetwarzania tekstów języka naturalnego [zob. Beelders, Plessis 2016: 10].

W jednych z pierwszych badań z wykorzystaniem okulografii, których celem była ocena wpływu kolorowania składni na efektywność analizy kodu (wyznaczenie wyniku działania programu), zaobserwowano, że podświetlanie składni skraca czas wykonania zadania oraz że efekt ten słabnie wraz ze wzrostem poziomu doświadczenia w programowaniu [Sarkar 2015: 56]. Podobny wynik odnotowano w badaniach, w których rejestrowano proces pisania i debugowania kodu programu – podświetlanie składni w czasie realizacji tych czynności znacznie zmniejszyło czas ich wykonania [Dimitri 2015: 67].

Metodologia badań

Cel badań

Analiza powyższej literatury oraz wyniki badań własnych opisane w pracy [Lach, Andrzejewska 2016: 59–88] dostarczyły przesłanek do sformułowania celu badawczego niniejszego artykułu. Celem tym jest weryfikacja hipotez o braku różnic w zakresie efektywności wykonania zadania oraz braku zróżnicowania wartości parametrów eye-trackingowych w procesie wyszukiwania błędów w kodach programów, pomiędzy kodem, w którym zastosowano i niezastosowano kolorowania składni.

Aparatura badawcza

Do badań wykorzystany został eyetracker iViewX Hi-Speed firmy Senso Motoric Instrument (SMI). Jest to aparat przeznaczony do prowadzenia badań nieinwazyjnych o częstotliwości próbkowania 500/1250 Hz, zaliczany do grupy wysokiej klasy urządzeń stacjonarnych mających zastosowanie głównie w warunkach laboratoryjnych. Stanowisko badawcze składa się z komputera, z poziomu którego zarządza się całym eksperymentem, monitora oraz eyetrackera. Konstrukcja urządzenia pozwala na stabilne utrzymywanie głowy w bezruchu, bez ograniczania pola widzenia badanego. Do przygotowania eksperymentu oraz opracowania jego wyników zastosowano programy firmy SMI: Experiment Center oraz BeGazeTM 2.4.

Uczestnicy badania i procedura

W badaniu udział wzięło 300 studentów kierunku informatyka na Uniwersytecie Pedagogicznym w Krakowie. Grupa składała się z 11 mężczyzn oraz 2 kobiet. Wszyscy studenci odbyli kurs programowania w języku C++. Każde badanie przeprowadzane było indywidualnie, a procedura badawcza rozpoczynała się od kalibracji i walidacji urządzenia, po czym uczestnikom prezentowana była plansza z instrukcją oraz kolejno slajdy z kodami kompletnych, ale krótkich programów napisanych w języku C++, w których osoby badane miały za zadanie zlokalizować błędy składniowe. Każdy program prezentował rozwiązanie tego samego problemu, zawierał wywołanie funkcji, która zwracała najmniejszą z 3 liczb będących jej argumentami.

```cpp
#include <iostream>

int minimum(int a1, int a2, int a3);

int main(void){
    std::cout << "Najmniejsza wartosc: " << minimum(3, 41, 7);
    return 0;
}

int minimum(int a1, int a2, int a3)
{

    int min;
    if(a1=<a2)  min=a1;
    else    min=a2;
    if(a3<min ) min=a3;

    return min;
}
```

```cpp
#include <iostream>

int min(int a, int b, int c){

    if( a < b && a < c )
        return a;
    if( b < a && b < c )
        return b;
    return c;

}
int main(void){

    std: cout << 'Najmniejsza wartosc: ' << min(61, 2, 13);

    return 0;
}
```

Rysunek 1. Kody analizowanych programów

W celu weryfikacji sformułowanych powyżej hipotez badawczych do analizy wybrano dwie spośród prezentowanych w czasie eksperymentu plansz (zob. rysunek 1: kod lewy – K1, kod prawy – K2).

Kody te zostały wybrane, ponieważ liczba wierszy i „zagęszczenie" tekstu programu na obu planszach były bardzo zbliżone. Głównym wyróżnikiem kodu K1 było kolorowanie składni. Skorzystano tutaj ze standardowego zestawu kolorów automatycznie nakładanego przez program DEV C++. Kod K2 był czarno-biały. W obu prezentowanych kodach umieszczono po dwa typowe błędy składniowe. W kodzie K1 był to brak klamry rozpoczynającej blok funkcji (błąd K1.1) oraz zamiana kolejności znaków w instrukcji warunkowej: „<=" (błąd K1.2), natomiast w kodzie K2 brak drugiego dwukropka po słowie std (błąd K2.1) oraz użycie znaków apostrofów zamiast cudzysłowów w instrukcji cout (błąd K2.2).

Wyniki badań i dyskusja

Efektywność wyszukiwania błędów okazała się wyższa w przypadku kodu K2, czyli czarno-białego, gdzie 54% i 31% badanych odszukało i opisało właściwie błędy K2.1 i K2.2. Ale w kodzie tym również więcej osób uznawało poprawne fragmenty jako błędne. W kolorowym tekście programu prawidłowo odnalazło błędy składniowe odpowiednio 54% (K1.1) i 8% studentów (K1.2). W teście chi-kwadrat (chi-k = 0,75; df = 1; p = 0,39) różnice w liczbie wskazanych i niewskazanych błędów pomiędzy kodami K1 i K2 nie okazały się istotne.

Jak można odczytać z tabeli 1, w przypadku kodu z kolorową składnią K1 zarówno czas jego analizy, jak i liczba fiksacji oraz średni czas trwania fiksacji i średnie amplitudy sakad miały wartości wyższe niż wartości analogicznych parametrów uzyskane dla kodu czarno-białego. Średnia wartość częstotliwości fiksacji była natomiast wyższa w przypadku kodu czarno-białego, co może być interpretowane jako przewaga procesów wyszukiwania nad procesami przetwarzania. Przeprowadzony test-t dla dwóch prób zależnych wykazał jednak, że różnice te dla wszystkich wymienionych parametrów nie były istotne statystycznie.

Tabela 1. Test t dla prób zależnych – wybrane parametry eye-trackingowe

Kod programu	K1		K2				
Parametr	Średnia	Odch. st.	Średnia	Odch. st.	T	df	p
Czas [ms]	88 743	54 262	64 771	33 949	1,78	12	0,10
Liczba fiksacji	309,85	206,67	220,69	101,13	2,01	12	0,07
Średni czas fiksacji [ms]	212,18	37,06	206,38	33,01	1,16	12	0,27
Częstotliwość fiksacji [liczba/s]	3,42	0,84	3,53	0,72	–1,38	12	0,19
Średnia amplituda sakad [°]	4,08	1,30	3,58	0,63	1,75	12	0,11

Natomiast istotne statystycznie (poziom istotności $p < 0,05$) – z wyjątkiem czasu analizy kodu ($r = 0,47$; $p > 0,05$) – okazały się współczynniki korelacji pomiędzy wartościami tych parametrów mierzonymi dla kodu K1 i K2. Wyniosły

one odpowiednio: dla liczby fiksacji r = 0,66, dla średniego czasu fiksacji r = 0,87, dla częstotliwości fiksacji r = 0,94, dla średniej amplitudy sakad r = 0,60. Zależności te są zatem silne i bardzo silne.

Na podstawie uzyskanych wyników można wnioskować, że badani niezależnie od prezentowanego kodu analizowali go w podobny sposób (właściwy dla danej osoby), stosując te same strategie przeszukiwania. Hipoteza o braku różnic w zakresie efektywności znajdowania błędów i wartości omówionych parametrów eye-trackingowych w procesie analizowania kodu kolorowego i czarno-białego została pozytywnie zweryfikowana.

Wyniki uzyskane w zakresie ogólnej liczby fiksacji oraz średniego czasu fiksacji, które nie okazały się istotnie różne w przypadku kodu czarno-białego i kolorowego, potwierdziły ustalenia wcześniejszych badań, że podświetlanie składni nie wpływa istotnie na wartości tych parametrów[Sarkar 2015: 56; Beelders, Plessis 2016: 10]. Podobnie jak w badaniach T.R. Beeldersa i J.-P.L. Plessisa średni czas trwania fiksacji był porównywalny do wartości charakterystycznych dla czytania tekstów języka naturalnego. W przeciwieństwie jednak do ustaleń Beeldersa i Plessisa w niniejszym badaniu wartości obu omawianych parametrów fiksacji w przypadku kodu kolorowego były wyższe. Prawdopodobnie było to skutkiem braku randomizacji w procesie wyświetlania plansz z kodami – kod kolorowy był analizowany przez wszystkich badanych jako pierwszy. Można przypuszczać, że na początku eksperymentu badani byli bardziej zaangażowani i stąd duże różnice w czasie i skorelowanej z nim liczbie fiksacji (K1: $r = 0,92$, $p < 0,05$; K2: $r = 0,85$, $p < 0,05$). Należy również wziąć pod uwagę fakt, że kody fragmentów programów, które analizowane były w badaniach Beeldersa i Plessisa, nie zawierały żadnych błędów i skupiono się tam wyłącznie na kolorowaniu składni. A typ błędu lub jego umiejscowienie to również potencjalne czynniki wpływające na uzyskane rezultaty.

Podsumowanie

Czytanie i interpretacja kodu programu to podstawowe kompetencje programistów, które, jak sugerują badania, mogą mieć związek z umiejętnością efektywnego rozwiązywania problemów, zatem rozwój tych zdolności powinien być składowym elementem praktycznej nauki programowania jako komplementarnej w stosunku do umiejętności pisania kodu. W związku z tym badania nad wpływem formatowania kodu na jego odczytywanie i rozumienie są istotne, a ich wyniki mają charakter aplikacyjny. Mogą mieć zastosowanie zarówno w dydaktyce informatyki – ustalono, że kolorowanie składni wpływa na analizę kodu przez początkujących programistów, jak i przy projektowaniu profesjonalnych środowisk programistycznych, a twórcy takich środowisk wskazują na niedostatek publikacji w tym zakresie. Zauważa się tutaj brak badań nad rolą parametrów fizjologicznych procesu widzenia, np. nad wrażliwością oka na

różne kolory [Jacques, Kristensson 2015: 29]. Eksperymenty takie mogłyby dostarczyć nowych informacji na temat wpływu podświetlania składni kodu na szeroko rozumianą efektywność programowania.

Literatura

Baecker R., Marcus A. (1986), *Design Principles for the Enhanced Presentation of Computer Program Source Text*, „Proceedings of CHI'86 Conference on Human Factors in Computing Systems".

Beelders T.R., Plessis du J.-P.L. (2016), *Syntax Highlighting as an Influencing Factor When Reading and Comprehending Source Code*, „Journal of Eye Movement Research" no. 9(1).

Binkley D., Davis M., Lawrie D., Maletic J.I., Morrell C., Sharif B. (2013), *The Impact of Identifier Style on Effort and Comprehension*, „Empirical Software Engineering" no. 18(2).

Dimitri G.M. (2015), *The Impact of Syntax Highlighting in Sonic Pi*, „Proceedings of the 26th Annual Conference of the Psychology of Programming Interest Group (PPIG 2015)".

Gomes A., Mendes A.J. (2007), *Learning to Program – Difficulties and Solutions*, „Proceedings of the International Conference on Engineering Education (ICEE 2007)", http://icee2007.dei.uc.pt/proceedings/papers/411.pdf (05.2016).

Jacques J.T., Kristensson P.O. (2015), *Understanding the Effects of Code Presentation*, „Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2015)".

Lach M., Andrzejewska M. (2016), *Wpływ formatowania kodów źródłowych programów na wykrywanie błędów składniowych – badania eyetrackingowe* [w:] W. Błasiak (red.), *Neuronauka i eyetracking. Badania i aplikacje*, Kraków.

Lister R., Adams S., Fitzgerald S., Fone W., Hamer J., Lindholm M., McCartney R., Moström J.E., Sanders K., Seppälä O., Simon B., Thomas L. (2004), *A Multi-National Study of Reading and Tracing Skills in Novice Programmers*, „SIGCSE Bulletin" vol. 36, issue 4.

Miara R.J., Musselman J.A., Navarro J.A., Shneiderman B. (1983), *Program Indentation and Comprehensibility*, „Communications of the ACM" no. 26(11).

Oman P.W., Cook C.R. (1990), *Typographic Style is More Than Cosmetic*, „ACM Communications" no. 33(5).

Rambally G. (1986), *The Influence of Color on Program Readability and Comprehensibility*, „Proceedings of 17th Technical Symposium on Computer Science Education (SIGCSE)".

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)".

Siegmund J., Kästner C., Apel S., Parnin C., Bethmann A, Leich T., Saake G., Brechmann A. (2014), *Understanding Understanding Source Code with Functional Magnetic Resonance Imaging*, „Proceedings of the 36th International Conference on Software Engineering (ICSE)".