ISSN 2355-0082

The Use of Color Gradation on Program Visualization for Learning Programming

Rossevine Artha Nathasya[1], Oscar Karnalim[2]
Faculty of Information Technology, Maranatha Christian University, Bandung, Indonesia
rossevine.artha@gmail.com
oscar.karnalim@it.maranatha.edu

Received on 15 February 2019
Accepted on 24 June 2019

***Abstract*—According to several works, Program Visualization (PV) enhances student understanding further about how a particular program works. However, to our knowledge, no PVs utilize color gradation as a part of their features, even though color plays an important role in visualization. Therefore, two uses of color gradation on PV are proposed on this paper. On the one hand, color gradation can be used to display execution frequency of each instruction; instruction with higher execution frequency will be assigned with more-prominent color. Such piece of information is expected to help student for understanding program complexity. On the other hand, color gradation can also be used to display access frequency of each variable; variable with higher access frequency will be assigned with more-prominent color. Such piece of information is expected to help student for understanding program-to-variable dependency. Both uses are proved to be effective for learning programming according to our evaluation.**

***Index Terms*—program visualization, color gradation, program complexity, program-to-variable dependency, computer science education**

I. INTRODUCTION

As the impact of Information Technology (IT) in daily life grows rapidly, programming becomes a promising skill to be learned; the demand of program development is increased. However, learning programming is not a trivial task; high logical thinking and clear understanding about abstractive concepts are required. As a result, several educational tools for learning programming are proposed [1].

Program Visualization (PV) is a programming-focused educational tool that helps the user to understand his/her source code (i.e. program) through visualization [1]. A typical PV works by accepting a code and then generating visualization states as its result (each state displays program information after a particular instruction has been executed). Using such tool, users are expected to understand their code further; resulted states are visualized in descriptive and interactive manner.

To our knowledge, no PVs utilize color gradation as a part of their features, even though color plays an important role in visualization. Hence, this paper explores the use of color gradation on PV. To be specific, color gradation will be applied for displaying two pieces of information: execution frequency of each instruction and access frequency of each variable. The execution frequency of each instruction is related to program complexity. With this information, the students can understand which parts of their program are executed most, and if they want to optimize, they know that those parts should be the main focus. The access frequency of each variable is related to program-to-variable dependency. With this information, the students can exclude unused variables. Further, if they want to optimize their code, variables with high access frequency can be addressed last; these variables are heavily related to the program and their optimization may take a considerable amount of time. We would argue that both pieces of information are important for users to learn their code at Introductory Programming course. They can get the main idea of program optimization prior taking the real material at more advanced courses.

II. RELATED WORKS

In general, educational tools for learning programming can be classified into twofold: standard educational tool and Software Visualization (SV). Standard educational tool enhances user understanding with no strong emphasis on visualizing software (i.e. program) data and process. For instance, a work proposed by [2] is more focused on comparing both algorithm and program time complexity in empirical manner. Other examples are the work proposed by [3,4]; they are primarily focused on active learning about Greedy algorithm. In contrast, SV enhances student understanding with a strong emphasis on visualizing software data and process [1]. Since software can be perceived from two perspectives, SV is usually classified further to two sub-categories: Algorithm and Program Visualization.

Algorithm Visualization (AV) focuses on visualizing algorithm (i.e. an abstractive representation of program). This kind of tool is frequently used to cover complex topics such as data structures and

algorithm strategies. First, a work in [5] proposes an AV for learning basic data structure material in accordance with an undergraduate course in a private university. Second, a work in [6] proposes a web-based portal for learning data structure and algorithm where more emphasis is given on data structure; the materials itself vary from the simplest one (e.g. stack) to the most complex one (e.g. graph). Third, a work in [7] proposes an AV for learning algorithm strategies by example; it covers brute force, dynamic programming, backtracking, and greedy algorithm strategy. Last, a work in [8] proposes an AV for learning branch \& bound strategy; it utilizes traveling salesperson problem as its case study.

Different with AV, Program Visualization (PV) focuses on visualizing program. Several examples of such tool are PythonTutor, Jelliot 3, Ville, Omnicode, and PITON. First, PythonTutor [9] is a web-based PV that is originally designed to teach user how Python program works. Second, Jelliot 3 [10] is a PV for learning Java program. Third, Ville [11] is a PV with language-independent design; new programming language can be incorporated directly as long as it has similar characteristic with the existing ones. Fourth, Omnicode [12] is an extended PV from PythonTutor; it introduces live programming environment. Fifth, PITON [13] is an integration between PV and programming workspace; where the user can visualize their code during the completion of their assessment.

Color gradation is a mechanism to gradually transition one color to another [14,15]. To our knowledge, this mechanism has not been used on any PVs even though color plays an important role in visualization.

III. METHODOLOGY

This paper proposes two uses of color gradation in Program Visualization (PV). One of them is about displaying execution frequency of each instruction while another one is about displaying access frequency of each variable. Both of them will be applied on CPyn (Colorized PYthon code visualizatioN), a prototype PV specifically developed to implement the uses of color gradation. As its name states, CPyn is specifically designed to visualize Python program. The user interface of CPyn can be seen on Figure 1; color gradation will be displayed in source code display (the middle component) and variable display (the upper-right component).

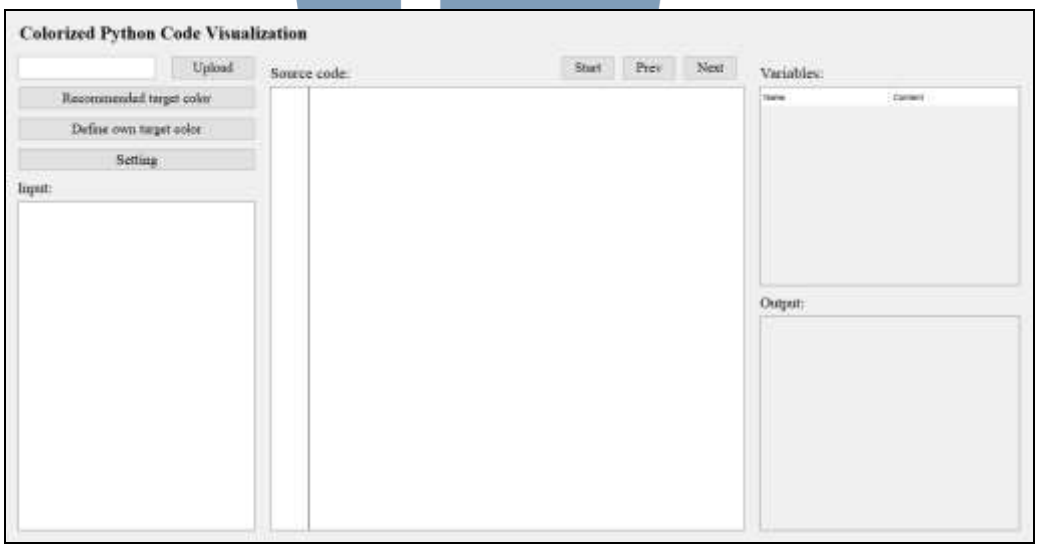

Figure 1. The user interface of CPyn

CPyn accepts a program source code file path, program input, and target color from left components. Target color can be defined by clicking either *Recommended target color* or *Define own target color* button. The former one will display popular colors to be selected while the latter one will display color picker where the user can input his/her own color. After all inputs are given, CPyn will generate visualization along with program output in step-by-step fashion; user can view each visualization state through *prev* and *next* button.

Execution frequency refers to how many times a particular instruction has been executed while running the program. Displaying such information is expected to provide further understanding about program complexity; instruction with higher complexity (i.e. instruction with higher execution frequency) will be displayed with more-prominent color. Since newline is a default instruction separator in Python (i.e. CPyn's target language), each instruction is assumed to occupy one line. Intuitively, displaying execution frequency is implemented in two typical PV phases: recording and visualization phase.

Execution frequencies will be recorded at recording phase (see Figure 2) by embedding two additional steps (the italicized ones). First, before conducting pseudo-execution, it will prepare an empty array to store execution frequencies. Second, according to executed instruction, it will increment given array content; the array will be then included as the result of recording phase.

Resulted array from recording phase will be displayed at visualization phase (see Figure 3) by embedding two additional steps (the italicized ones). First, before visualization starts, it will define the highest frequency as maximum threshold according to resulted array. This maximum threshold is required to define the most prominent color in gradation equation. Second, during visualization, the background of each line will be recolored based on resulted color gradation. To keep instruction text still readable, foreground color of each instruction text will be inverted toward its background color.

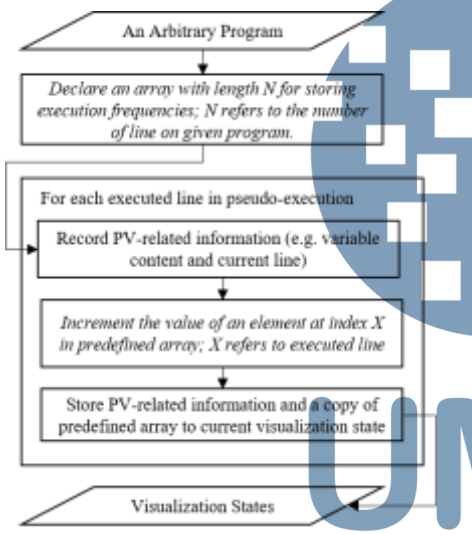

Figure 2. Recording Phase

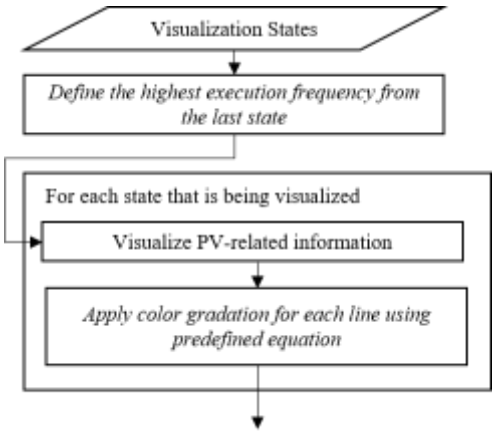

Figure 3. Visualization Phase

In order to generate color gradation, two color equations are proposed according to [15]: RGB-based and CMYK-based equation. Both of them accept three parameters per instruction: execution frequency (*ef*), maximum frequency threshold (*mf*), and target color (in either RGB or CMYK format). User can select one of these equations at CPyn's setting.

RGB-based color equation generates the color of each instruction by combining red, green, and blue color component from (1), (2), and (3) respectively; *R'*, *G'*, and *B'* are the color components of target color. Since black is RGB's default color where all components are assigned as 0, resulted gradation will be assigned from black to target color (higher frequency refers to brighter color). An example of CPyn's source code display resulted from such gradation can be seen on Figure 4. It becomes dark-themed; RGB-based color equation uses black as its initial color.

$$R = (R'*ef)/mf \quad (1)$$

$$G = (G'*ef)/mf \quad (2)$$

$$B = (B'*ef)/mf \quad (3)$$

In contrast, CMYK-based color equation generates the color of each instruction by combining cyan, magenta, yellow, and black color component from (4), (5), (6), and (7) respectively; *C'*, *M'*, *Y'*, and *K'* are the color components of target color. Different with RGB-based color equation, resulted gradation will be assigned from bright to target color (higher frequency refers to darker color); white is CMYK's default value where all components are assigned as 0. An example of CPyn's source code display resulted from such gradation can be seen on Figure 5. It becomes bright-themed; CMYK-based color equation uses white as its initial color.

$$C = (C'*ef)/mf \quad (4)$$

$$M = (M'*ef)/mf \quad (5)$$

$$Y = (Y'*ef)/mf \quad (6)$$

$$K = (K'*ef)/mf \quad (7)$$

Access frequency refers to how many times a particular variable has been accessed while running the program. Displaying such information is expected to provide further understanding about program-to-variable dependency; variable that is more depended by given program (i.e. variable with higher access frequency) will be displayed in more-prominent color.

Displaying access frequency is implemented in similar manner as displaying execution frequency. It only differs in four aspects. First, prepared data container at recording phase is replaced with a key-value pair set where key refers to variable name and value refers to its frequency (in this work, we assume variables are distinguishable through its name). Second, update mechanism at recording phase will be conducted by comparing the value of recorded variables from current state with its adjacent previous

state. For each variable, if its value is changed, its access frequency on prepared data container will be incremented by 1. Third, the highest frequency at visualization phase will be defined based on prepared key-value pair set. Last, color gradation will be displayed on variable display instead of source code display (see Figure 6 for an example of CPyn's variable display resulted from RGB-based gradation and Figure 7 for an example of CPyn's variable display resulted from CMYK-based gradation). Resulted color for each variable entry refers to how many times that variable has been accessed from initial to current visualization state; brighter color refers to higher frequency.

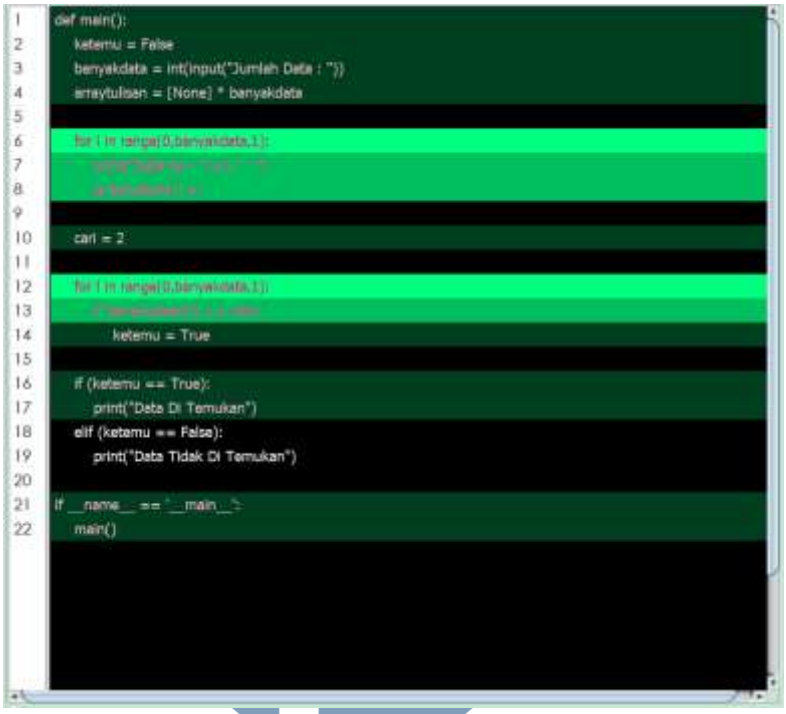

Figure 4. An example of CPyn's source code display resulted from RGB-based color gradation

Figure 5 An example of CPyn's source code display resulted from CMYK-based color gradation

IV. EVALUATION

In order to evaluate proposed uses about color gradation, two evaluation scenarios are conducted: functionality-based and questionnaire-based evaluation. Functionality-based evaluation validates whether proposed uses are implemented correctly. Each implementation is evaluated by comparing its resulted color gradation and frequencies with manually-calculated result. Seven source codes are considered per implementation; each code covers different introductory programming materials. According to our functionality-based evaluation, both uses are implemented correctly; their resulted color gradation and frequencies is similar to manually-calculated result on all codes.

Figure 6. An example of CPyn's variable display resulted from RGB-based color gradation

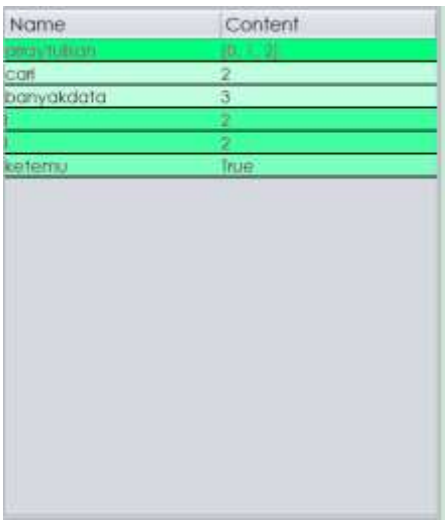

Figure 7. An example of CPyn's variable display resulted from CMYK-based color gradation

Different with functionality-based evaluation, questionnaire-based evaluation validates whether proposed uses are useful in practice from human perspective. It involves 20 undergraduate students where each user (i.e. student) is asked to answer 11 questions related to practical values of both uses. Before answering these questions, each user is required to complete 30 problems related to execution and access frequency using CPyn in 30 minutes; each problem is related to introductory programming material and its expected solution is only about one to three words. Such prerequisite aims to provide a real experience for users about both uses when learning programming.

Questions used in this survey are classified into three categories: scale-based, feedback, and bug report question. Scale-based question asks about user agreement toward predefined statement in 7-points Likert scale (1 refers to completely disagree, 4 refers to neutral, and 7 refers to completely agree). Nine questions fall into this category where their question ID and statement can be seen on Table 1.

Table 1. Statements involved in scale-based questions

ID	Statement
Q1	Color gradation in source code display helps the user to determine which instruction is either the most or the least frequently-executed one.
Q2	Color gradation in variable display helps the user to determine which variable is either the most or the least frequently-accessed one.
Q3	Color gradation in source code display helps the user to determine non-executed instructions regarding to a particular input set.
Q4	Color gradation in variable display helps the user to determine non-accessed variable regarding to a particular input set.
Q5	Color gradation in source code display, at some extent, helps the user to understand program complexity.
Q6	Dark-themed color gradation (i.e. RGB-based color gradation) in source code display is convenient to be used.
Q7	Bright-themed color gradation (i.e. CMYK-based color gradation) in source code display is convenient to be used.
Q8	Dark-themed color gradation (i.e. RGB-based color gradation) in variable display is convenient to be used.
Q9	Bright-themed color gradation (i.e. CMYK-based color gradation) in variable display is convenient to be used.

According to the result of scale-based questions (see Figure 8 where vertical axis represents resulted scale and horizontal axis represents the questions), all statements are positively agreed; they are assigned with mean score higher than 4 (tend to positive). In other words, it can be stated that color gradation is considerably beneficial when used as a part of PV features. Among these statements, Q1 is assigned with

the highest mean (6.25 of 7); the difference between the most and the least frequently-executed instruction is shown clearly in contrast coloring. It is true that such contrast coloring is also found on variable display (referring to Q2 statement). However, according to respondents' informal feedback, program-to-variable dependency (i.e. Q2's target information) is less useful than program complexity (i.e. Q1's target information).

Q8 is assigned with the lowest mean (4.7 of 7) when compared to other statements. Further observation shows that dark-themed gradation causes some texts on variable display are unreadable. It is true that such issue should also be found on source code display (referring to Q6 statement). However, according to respondents' informal feedback, text on source code display is rarer to be read than text on variable display when learning through visualization.

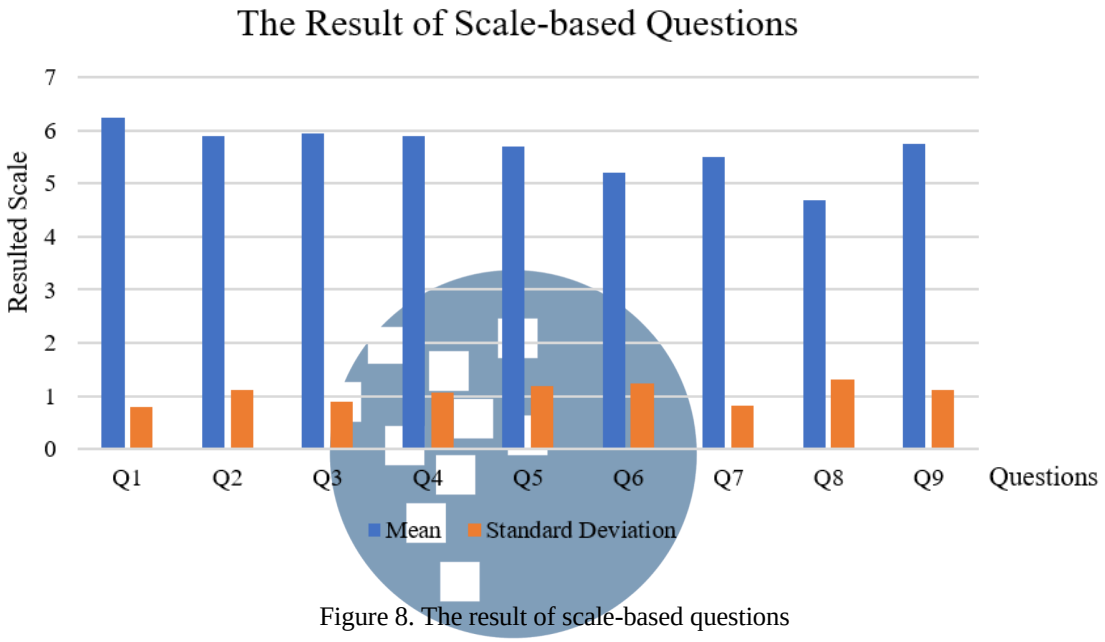

Figure 8. The result of scale-based questions

When perceived from the variability (i.e. standard deviation) of resulted means, Q1 is assigned with the least-varied result (0.786). Hence, it can be stated that all respondents share similar perspective about Q1; they strongly agree that color gradation in source code display helps the user to determine which instruction is either the most or the least frequently-executed one. On the contrary, Q8 is assigned with the most-varied result (1.301). In other words, it can be stated that not all respondents share similar perspective about Q8. Some of them do not slightly agree that dark-themed color gradation in variable display is convenient to be used.

Feedback question is an open-ended question asking about respondents' feedback about proposed uses of color gradation. Generally speaking, most respondents only strengthen their answers on scale-based questions (e.g. reclaiming that color gradation helps the user to determine which instruction is either the most or the least frequently-executed one). Only one respondent provides different answer. He states that displayed font should be bigger for high readability. We will solve such issue on future work by providing resize-able font as a feature.

Bug report question is an open-ended question asking about bugs found by respondents while trying the prototype application (i.e. CPyn). Two bugs are found which are about reset button that does not work and variable display that shows inconsistent content on a particular occasion. Both bugs have been fixed at the time of writing this paper.

V. Conclusion and Future Works

In this paper, two uses of color gradation as a part of PV's features have been proposed. One of them is related to program complexity while the another one is related to program-to-variable dependency. According to our evaluation using CPyn (i.e. a prototype PV specifically designed to implement such uses), both uses are effective to help students for learning programming.

For future work, we plan to evaluate proposed uses based on students' grade in real courses. To be specific, a quasi-experimental design [16] will be used to compare students' grade on two materials: time complexity in Algorithm Strategy course and program optimization in Competitive Programming course. The first material is related to program complexity while the latter one is related to program-to-variable dependency.

REFERENCES

[1] Sorva J, Juha. Notional machines and introductory programming education. ACM Transactions on Computing Education 2013;13:1–31. doi:10.1145/2483710.2483713.

[2] Elvina E, Karnalim O. Complexitor: An Educational Tool for Learning Algorithm Time Complexity in Practical Manner. ComTech: Computer, Mathematics and Engineering Applications 2017;8:21. doi:10.21512/comtech.v8i1.3783.

[3] Velázquez-Iturbide JÁ, Pérez-Carrasco A. Active learning of greedy algorithms by means of interactive experimentation. Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education - ITiCSE '09, vol. 41, New York, New York, USA: ACM Press; 2009, p. 119. doi:10.1145/1562877.1562917.

[4] Debdi O, Paredes-Velasco M, Velázquez-Iturbide JÁ. GreedExCol, A CSCL tool for experimenting with greedy algorithms. Computer Applications in Engineering Education 2015;23:790–804.

[5] Christiawan L, Karnalim O. AP-ASD1: An Indonesian Desktop-based Educational Tool for Basic Data Structure Course. Jurnal Teknik Informatika Dan Sistem Informasi 2016;2.

[6] Halim S, Chun KOH Z, Bo Huai LOH V, Halim F. Learning Algorithms with Unified and Interactive Web-Based Visualization. Olympiads in Informatics 2012;6:53–68.

[7] Jonathan FC, Karnalim O, Ayub M. Extending The Effectiveness of Algorithm Visualization with Performance Comparison through Evaluation-integrated Development. Seminar Nasional Aplikasi Teknologi Informasi (SNATI), 2016.

[8] Zumaytis S, Karnalim O. Introducing an Educational Tool for Learning Branch & Bound Strategy. Journal of Information Systems Engineering and Business Intelligence 2017;3:8. doi:10.20473/jisebi.3.1.8-15.

[9] Guo PJ. Online python tutor: embeddable web-based program visualization for cs education. Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13, New York, New York, USA: ACM Press; 2013, p. 579. doi:10.1145/2445196.2445368.

[10] Moreno A, Myller N, Sutinen E, Ben-Ari M. Visualizing programs with Jeliot 3. Proceedings of the working conference on Advanced visual interfaces - AVI '04, New York, New York, USA: ACM Press; 2004, p. 373. doi:10.1145/989863.989928.

[11] Rajala T, Laakso M-J, Kalla E, Salakoski T. VILLE: a language-independent program visualization tool. Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88, Darlinghurst: Australian Computer Society; 2007, p. 151–9.

[12] Kang H, Guo PJ. Omnicode: A Novice-Oriented Live Programming Environment with Always-On Run-Time Value Visualizations. The 30th ACM Symposium on User Interface Software and Technology (UIST, 2017.

[13] Elvina E, Karnalim O, Ayub M, Wijanto MC. Combining program visualization with programming workspace to assist students for completing programming laboratory task. Journal of Technology and Science Education 2018;8:268. doi:10.3926/jotse.420.

[14] Martin J (Frances J. The encyclopedia of pastel techniques. Running Press; 1992.

[15] Anderson FL. Fabric to Dye For: Create 72 Hand-Dyed Colors for Your Stash; 5 Fused Quilt Projects. 2010.

[16] Creswell JW. Educational research : planning, conducting, and evaluating quantitative and qualitative research. Pearson; 2012.