Visualization of program development process

Conference Paper · September 2019

DOI: 10.1109/STC-CSIT.2019.8929774

2 authors:

Oleksandr Zhevaho
Dnipropetrovsk National University of Railway Transport
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

V. I. Shinkarenko
Dnipropetrovsk National University of Railway Transport
67 PUBLICATIONS **44** CITATIONS
SEE PROFILE

© 2019 IEEE
CSIT 2019, 17-20 September, 2019, Lviv, Ukraine
https://doi.org/10.1109/STC-CSIT.2019.8929774

Visualization of program development process

Viktor Shynkarenko
Department of Computer Information Technology
Dnipropetrovsk National University of Railway Transport
named after academician V. Lazaryan
Dnipro, Ukraine
shinkarenko_vi@ua.fm

Oleksandr Zhevago
Department of Computer Information Technology
Dnipropetrovsk National University of Railway Transport
named after academician V. Lazaryan
Dnipro, Ukraine
marakonec@gmail.com

***Abstract*—The aim of this paper is to improve the process of program development using high-level languages. The practical application in programming training is expected, especially at early stages. The teacher gets an opportunity of visual monitoring of the program development process, which means the active participation in the formation of an effective student programming style. The extension for Visual Studio, which monitors the process of programming, is developed.**

Keywords—stepwise refinement method, visual studio extension, program development process, visualization, programming training, programming style.

I. Introduction

One of the urgent problems of the educational process in the field of information technologies is the low level of practical skills of students' in the development of computer programs.

Modern training methods allow you to monitor and evaluate only the final result, in the form of a running program and a report with the text of the program, instead of the program development process. Therefore, such an important part as the process of writing a program remains without proper attention.

There are a number of software tools for analyzing the quality of program texts, but they analyze the finished texts only, and there are no such tools for analyzing the programming style.

In the process of learning the basics of programming, it is important to identify the problems and assist in their elimination, to control the independence and quality of performance of tasks, to record the difficulties in the work of each student, to reveal his/her hidden opportunities in development of program texts. It requires the constant supervision of students by the teacher. However, it is practically impossible to monitor the individual features of the programming process of each student during group classes.

In order to implement an opportunity to measure the characteristics of style of each student both in the classroom and during individual work, we propose the automation of this process with the use of specially designed extension to the Visual Studio.

II. Stepwise refinement method

The high level of practical training in programming implies the availability of effective style of the developer in the process of developing algorithms and programs. It includes: usage of the method of stepwise refinement, decomposition skills, and compliance with programming standards.

The principle of stepwise refinement was introduced by Niklaus Wirth, who published the detailed article with the example of using this principle in Communications of ACM Journal in 1971 [1].

Stepwise refinement is the paradigm of developing a complex program from simple, with the gradual addition of parts [2, 3]. Programs must be written for people to read, and only incidentally for machines to execute [4].

The deviation from the stepwise refinement method is expressed in chaotic writing of the program text, making changes to already written text fragments, and inability to add comments timely. It should be noted that there are approaches to execution of programs which do not involve the active use of comments [5].

At the same time, such technologies as the technology of literate programming of D. Knuth [6] suggest the active use of comments where comments actually become the documentation attached to the program.

The usage of decomposition skills is characterized by the order of creation of new functions, ratio of function sizes, their average size [7].

III. Methods for obtaining the information on program development process

One of the reasons for observing the development process of the algorithm and text of the program by students is the monitoring of independence in the performance of work. The paper [8] proposes video-recording of the development process in combination with an explanation and stepwise description of all actions performed. But this approach is rather time consuming one for both the student and the teacher.

In recent years there have been several reports of successful use of Version Control Systems (VCS) in the classroom programming [9]. However, the version control system is not able to provide a comprehensive evaluation the actual contribution of a student to the final result, since it does not give the information on the process of development of the program text. VCS only captures the history of code changes between commits.

Today the only opportunity to analyze the style of work is direct monitoring of the development process, but the teacher is not able to control all students during classes. In order to implement the ability to measure the characteristics of each student's style during classes and individual work, we consider the automation of such process through a specially designed extension to the software development environment.

It is important that this tool will help to timely identify the problems in the process of training of each student by comparing with others, to reveal weaknesses in knowledge and skills, and to help in their elimination. The collection of changes occurring in the process of writing the program text will help in analyzing the style.

One of the most common software development environments is Microsoft Visual Studio. Using the system to analyze the quality of the coding process in Visual Studio is very convenient, because it is widely used both in production and for training purposes.

The Visual Studio development environment is built on the principles of automation and scalability, allowing developers to integrate almost any new elements and interact with standard and custom components [10].

IV. Constructive complexity of program and process of development

Consider a computer program as a construction, and the process of its development as a constructive process.

The elements of the program are lexemes with semantic attributes, such as a reserved word, identifier, separator, operation sign, relation sign, and comment. They are interconnected by many different syntactic and semantic relations.

Based on the relations of aggregation, aggregation operations are performed (mentally, in a person's head). As a result, intermediate forms with independent syntactic and semantic meaning are formed.

Syntactic relations: following, immediately following, nesting, aggregation and inverse to them. Semantic relationships are usually N-ary. So, the comment has value if all the lexemes are present and in a certain order.

Even small programs have an extremely complex design. Only immediately following relations k-1, where k is the number of elements in the program, and followings (k-1)!, not to mention the great variety and number of other relations.

Consider a small example of an intermediate form – *numbers[mid]*. The two lexemes «[» and «]» are in syntactic relations of following and aggregating into a composite outfix unary relation. Semantically, *[mid]* defines the element number in the *numbers* array. The result is a semantic-syntactic aggregate – a term.

The task of developing a program (of a complex construction), of its text is considerably simplified if the writing of semantic-syntactic aggregates (terms, operators, semantically related groups of operators) is not broken. Such aggregates must be spelled out completely. On the one hand, the method of stepwise refinement provides for such an approach. On the other hand, the visualization of the program development process allows you to monitor the continuity of the development of aggregates.

V. Description of developed extension

To monitor the program development process, an extension for Visual Studio was created. Visual Studio provides an extensible, project-independent object model that presents solutions, projects, code objects, documents, and so on. Each type is represented by its corresponding automation interface.

The developed extension consists of three modules: storing of the program development history; calculating the difference between versions; visualization of development history.

The module for storing versions of the program development ensures that all versions of the update of the program texts are saved, except for those that do not contain useful information. It makes no sense to store all versions of the writing of each character in a line of text.

If keep the first or the latest version and patches, the consistent application of which will give new versions of the program, then the calculation of the difference between the versions can create the effect of "braking" the development environment.

Based on the analysis of the mechanisms for calculating the difference between versions and the available experimental data, the Histogram algorithm was chosen [11].

The module for storing the history of writing a program text is an implementation of the *IWpfTextViewCreationListener* interface. This allows you to handle all the events of the text editor in which the code is written.

The module saves not only the current version, but also information about changes to the text. Changes can be single-line and multi-line. If the change is single line then the line number is stored. Type of change can be: Insert, Edit, and Delete.

The module for calculating the difference between versions handles a collection of program versions with information about changes. The result is a sequence of continuously typed strings of characters. If the change is one-line, calculating the difference only for this line. If the type of change *Insert* does not need to do any calculations. Since this line will be the result. In other cases, the Histogram algorithm is used to calculate the difference between versions.

The visualization module uses Three.js [12] to demonstrate the development history of the program as a set of 3D cubes.

The input of this module is a dictionary. The key of the dictionary is the sequence number of the string added to the program being studied, the value is the string itself. Its output – sequence of 3D parallelepipeds of *n-i* height, where *n* –

quantity of strings in the dictionary, *i* – key of the dictionary. Consequently, the string written the first will have the maximum height.

VI. Results

Consider an example of visualization of the program development process. There is a text of the program:

```
 /// Searches the entire sorted array of numbers for an
    element.
/// <param name="numbers">Array of
numbers</param>
/// <param name="item">The object to locate.</param>
/// <returns>The index of the element if item is found,
        otherwise a negative number</returns>
public static int BinarySearch(int[] numbers, int item)
{
    int low = 0;
    int high = numbers.Length - 1;

    while (low <= high)
    {
        // Find the middle of the array
        int mid = low + (high - low) / 2;

        if (item < numbers[mid])
        {
            // If the search item is less than the value in the
               middle, then the high limit will be the element
               to the middle.
            high = mid - 1;
        }
        else if (item > numbers[mid])
        {
            // If the search key is greater than the value in the
               middle, then the lower limit will be the element
               after the middle.
            low = mid + 1;
        }
        else
        {
            // Item index found.
            return mid;
        }
    }
    // Item index not found.
    return -1;
 }
```

It history will look like as follows:

#1 – *public static int BinarySearch(int[] numbers, int item)*

#2 – *int low = 0;*

#3 – *int high = numbers.Length - 1;*

#4 – *return -1;*

#5 – *while (low <= high)*

#6 – *int mid = (high - low) / 2;*

#7 – *return mid;*

#8 – *if (item < numbers[mid])*

#9 – *high = mid - 1;*

#10 – *else if (item > numbers[mid])*

#11 – *low = mid + 1;*

#12 – *else*

#13 – */// Searches the entire sorted array of numbers for an element.*
/// <param name="numbers">Array of numbers</param>
/// <param name="item">The object to locate.</param>
/// <returns>The index of the element if item is found, otherwise a negative number</returns>

#14 – *// Item index not found.*

#15 – *// If the search item is less than the value in the middle, then the high limit will be the element to the middle.*

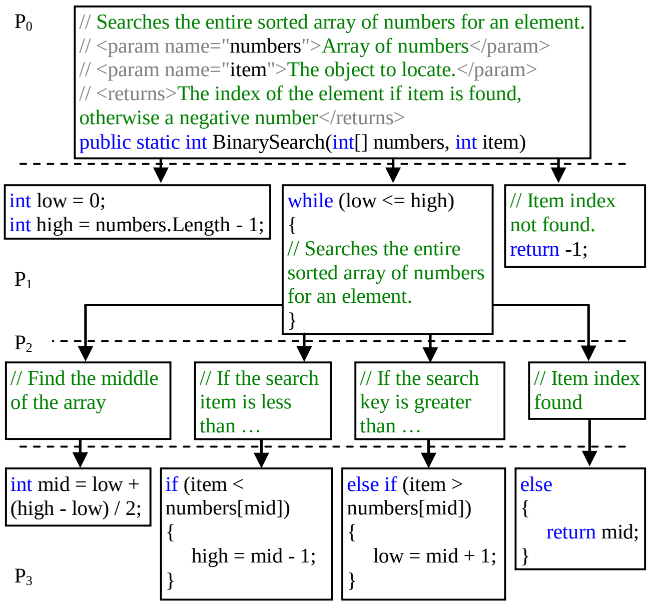
#16 – *// If the search key is greater than the value in the middle, then the lower limit will be the element after the middle.*

#17 – *low +*

#18 – *// Find the middle of the array*

#19 – *// Item index found.*

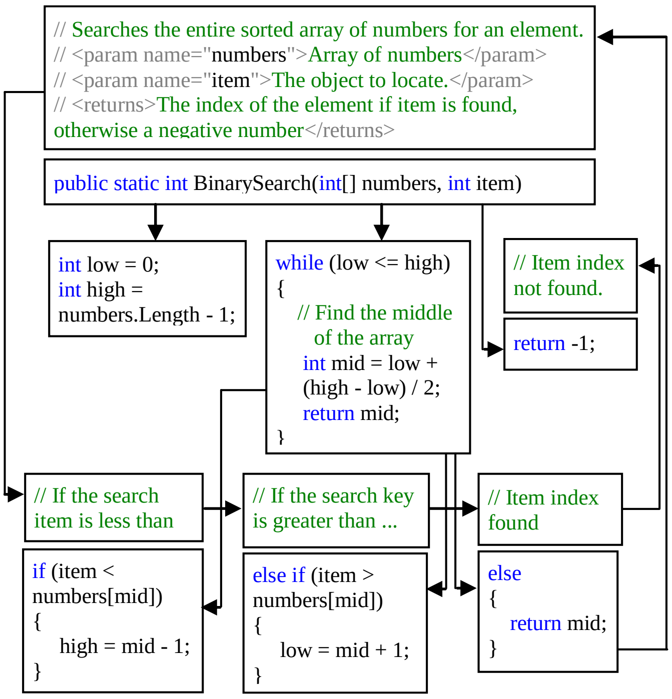
The iterative development process is presented in the Fig. 1. Any complex task is detailed at next iteration on next level P_i.



P_0
// Searches the entire sorted array of numbers for an element.
// <param name="numbers">Array of numbers</param>
// <param name="item">The object to locate.</param>
// <returns>The index of the element if item is found, otherwise a negative number</returns>
public static int BinarySearch(int[] numbers, int item)
P_1
int low = 0;
int high = numbers.Length - 1;
while (low <= high) { // Searches the entire sorted array of numbers for an element. }
// Item index not found. return -1;
P_2
// Find the middle of the array
// If the search item is less than …
// If the search key is greater than …
// Item index found
P_3
int mid = low + (high - low) / 2;
if (item < numbers[mid]) { high = mid - 1; }
else if (item > numbers[mid]) { low = mid + 1; }
else { return mid; }

Fig. 1. Graph representation of the program development process according to stepwise refinement method

An example of a typical deviation from the method of stepwise refinement of the program development process is presented on Fig. 2. This deviation is expressed in the chaotic writing of the program text, making changes to an already written piece of text, and not adding comments in appointed time.



```
// Searches the entire sorted array of numbers for an element.
// <param name="numbers">Array of numbers</param>
// <param name="item">The object to locate.</param>
// <returns>The index of the element if item is found,
otherwise a negative number</returns>
```

```
public static int BinarySearch(int[] numbers, int item)
```

```
int low = 0;
int high =
numbers.Length - 1;
```

```
while (low <= high)
{
   // Find the middle
      of the array
    int mid = low +
    (high - low) / 2;
    return mid;
}
```

```
// Item index
not found.
```

```
return -1;
```

```
// If the search
item is less than
```

```
// If the search key
is greater than ...
```

```
// Item index
found
```

```
if (item <
numbers[mid])
{
    high = mid - 1;
}
```

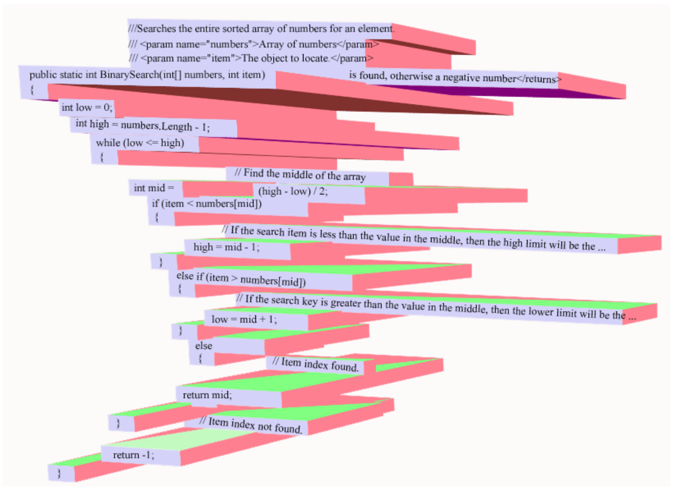
```
else if (item >
numbers[mid])
{
    low = mid + 1;
}
```

```
else
{
    return mid;
}
```

Fig. 2. Graph representation of the program development process with typical deviation from the stepwise refinement method

Visual 3D representation of development process allows the teacher receive all the necessary information in the convenient view.

3D representation of the coding process is provided (Fig. 3). Provides the ability to scale and rotate the 3D image.



```
///Searches the entire sorted array of numbers for an element
/// <param name="numbers">Array of numbers</param>
/// <param name="item">The object to locate.</param>
public static int BinarySearch(int[] numbers, int item)      is found, otherwise a negative number</returns>
{
    int low = 0;
    int high = numbers.Length - 1;
    while (low <= high)
    {
            // Find the middle of the array
        int mid =        (high - low) / 2;
        if (item < numbers[mid])
        {
            // If the search item is less than the value in the middle, then the high limit will be the ...
            high = mid - 1;
        }
        else if (item > numbers[mid])
        {
            // If the search key is greater than the value in the middle, then the lower limit will be the ...
            low = mid + 1;
        }
        else
        {
            // Item index found.
            return mid;
        }
    }
    // Item index not found.
    return -1;
}
```

Fig. 3. Visual 3D representation of the coding history

The visualization of the development process of the presented program clearly shows that the method of stepwise refinement is broken, because that the comments was written after the program text, and operator (aggregate) *int mid = low + (high - low) / 2;* was not written continuously.

Conclusions

Development style monitoring methods are the basis for raising the level of students practical training, reducing the time used inefficiently in the process of program development by the student and performance control by the teacher. The proposed tools will be useful for students in self-control of the acquired knowledge and skills in program development.

Developed tools comprise three parts. Functionality of the first part consists in obtaining of information on the order of making changes to the program text; the second part is aimed at preparation and analysis of the data obtained; purpose of the third part is in visual display of the relevant reports.

This software product provides the teacher with the information about each student work style, and quality of style with indication of individual features, thus helping to identify the time-consuming difficulties and to analyze the changes in the approach to program development during the period of training. Furthermore, the teacher will get an opportunity to specify and to verify compliance with certain requirements to the work, which can be checked in the process of development only.

References

[1] N.Wirth, “Program development by stepwise refinement”, Comm. ACM 14, 1971, pp.221–227.

[2] E.W.Dijkstra, “A Discipline of Programming”, Prentice-Hall, 1976.

[3] N.Gehani, “Program development by stepwise refinement and related topics”, The Bell System Technical Journal, 1981, pp.347–378.

[4] H.Abelson, G.J.Sussman and J.Sussman, “Structure and Interpretation of Computer Programs second edition”, Cambridge MA, 2011.

[5] K.Beck, “Extreme Programming Explained: Embrace Chage, second ed.”, Addison-Wesley, 2004.

[6] D.Knuth, “Literate programming”, Computer Journal, 27 (2), 1984, pp.97–111.

[7] S.McConnell, “Code complete, 2nd ed”, Microsoft Press, Redmond, 2004.

[8] J.Bennedsen and M.E.Caspersen, “Revealing the programming process”, Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, 2005, pp.186–190.

[9] A.Shtanyuk and D.Shagbazyan, “Version Control System in education process”, № 12 (26), 2017, pp.29.

[10] “Automation model overview”, [Online]. Available: https://docs.microsoft.com/ru-ru/visualstudio/extensibility/internals/automation-model-overview. [Accessed: 30 March 2019].

[11] Nugroho Y. S., Hata H. and Matsumoto K. (2019), "How Different Are Different diff Algorithms in Git? Use --histogram for Code Changes", arXiv:1902.02467.

[12] “Three.js fundamentals”, [Online]. Available: https://threejsfundamentals.org/threejs/lessons/ru/threejs-fundamentals.html. [Accessed: 30 March 2019].