2011 International Conference on e-Education, Entertainment and e-Management

3De - synergetic program visualization: a visual learning-aid tool for novice students

Affandy[1,2], Nanna Suryana[1], Sazilah Salam[1], Mohd Sanusi Azmi[1], and Edi Noersasongko[2]

[1]Faculty of Information & Communication Technology Universiti Teknikal Malaysia Melaka
Melaka, Malaysia
[2]Faculty of Computer Science University of Dian Nuswantoro
Semarang, Indonesia
[1]{affandy_ra, nsuryana, sazilah,sanusi}@utem.edu.my
[2]{affandy, edi-nur}@dosen.dinus.ac.id

***Abstract*— coding program is a dominant activity in programming cycle, nevertheless the equal attention must also given upon designing and evaluating the program in order to understanding the whole of the programming tasks. Unfortunately, these issues have not been addressed by current software visualization as a learning-aid tool. The development of 3De intended to help novice students to comprehend the multi forms of program from a higher level to a lower level within integrated environment. The basic idea of the system is to show students about the programming stages starting from designing problem-solving, developing code and validating logical flow of the program through visualizing multi level of program abstraction. Students preferred to construct their problem-solving visually, and 3De will simulate it to explain the program behaviour and data changing. A line-by-line auto generated syntax in C++ form will be produced accordingly and can be run in conventional C++ program developer**

Keywords: visualization; problem solving; programming; synergetic; learning-aid

I. INTRODUCTION

Learning to program is considered as a complex process that requires in practical manner both cognitive learning in conceptual perspective and skill performance. Under the light of conceptual implies that learner need to understand the declarative nature of programming knowledge, i.e. data type, syntax and semantic of particular programming language, algorithm or problem solving strategies, programming paradigm, control structure testing and etc. Furthermore, in skill performance perspective novice students also must have the ability to use or apply those conceptual of programming knowledge in programming related tasks such as designing the problem solving, transforming the designed algorithm into a computer code and verifying the validity of the program. There is a mutual dependency between both of them; and tends to be an overlapping among programming related tasks and also between conceptual and practical aspect, none of them should be take apart from the others [1].

In order to ease learning process of computer programming, the role of software engineering technology has been involving in a number of ways for decades. One key solution that has arisen is Software Visualization (SV), which is intended to represent information about software in graphical way using various visualization techniques and cognitive approaches. There are several types of visualization tools for software comprehension such as Program Visualization (PV), Algorithm Visualization (AV), Algorithm Animation (AA) and Data Visualization as well. Most of them focused on code development and visual explanation of algorithm, rarely tools which is support to algorithm development or algorithm-to-program translation.

There is a belief among the researchers that SV gives students better understanding of basic programming. Unfortunately, contradictory things happened on that belief, other more studies have shown the downfall of this technology as learning or teaching aid-tool [2-5]. Reiss stated that paradoxically euphoria of the SV caused by the lack of support to the realities of software understanding and development [6]. The fact of program understanding involves with all understanding programming related tasks, not just rely on one stage of programming, and these issues has not been addressed by current SV. Comprehensive study by Naps et.al, suggested that the learners' engagement in active learning process is more important than the sophisticated technology of SV [7].

Comprehensive review by Robins et.al [8] showed clearly that the understanding of knowledge and skill in computer programming are formed through the interrelated of two dimensions, namely the individual attributes dimension and the phase of programming cycle dimension. Even though developing or coding program be a dominant activity in programming cycle, nevertheless the equal attention must also given upon designing and evaluating the program. Von and Vans [9] emphasize on how the cognitive process of programmer on program understanding, since the computer program has a multidimensional representation, a whole understanding of programming knowledge including the programming language, application domain, algorithms, control-flow, data-flow etc are required to build the current internal representation (mental model) of the program.

3De is a synergetic program visualization tool providing multi abstract views of programming that support for design, development and debug for program comprehension. It has a build-in C++ syntax generator with which user can use its output to get the ready-to-run program code that compatible with C++ program developer. The visualization can be viewed in three levels of abstractions to support the novice

978-1-4577-1382-8/11/$26.00 ©2011 IEEE

students' comprehension. To emphasize the program process, 3De has a simultaneously view in displaying a program in higher-level of algorithm abstraction, intermediate-level of algorithm behaviour, and lower level of algorithm-code translation. To make visualization more interactive each part of the designed problem-solution will be executed immediately no need to wait for the complete structure of solutions designed.

The structure of this study is organized as follows. First, in section 2 we recognize some previous related works and tools. 3De synergetic program visualization tool and its key features are presented in section 3. In section 4, we discuss the potential effects of the system to the learning process and finally section 5 present the conclusion in brief.

II. RELATED PREVIOUS WORKS

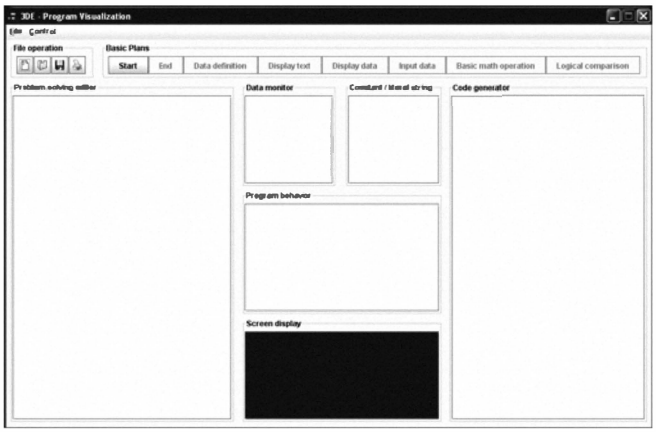
Learning programming is more than simply learning programming languages, nevertheless, typical introductory programming courses mostly adopt the learning material based on the programming textbooks, which mainly focus in the knowledge of particular programming language [10-12]. This situation leads to an ill-defined concept that programming equal to coding, writing texts of symbolic arrangement of data or instruction base on particular syntax of programming language. However, some of multi-national studies concluded that programming is close to the problem-solving tasks, learning programming means learners should be able to 1) Abstract the problem from its domain. 2) Break down the problems into sub-problems. 3) Construct sub-solution for each sub-problem. 4) Transforms each sub-solution into a working program, and 5). Evaluate and refine all the elements [10], [13]. It means that mutual dependency among designing, developing and debugging are things that cannot be denied. Furthermore, program is a very complex multidimensional representation, rather than textual syntaxes. It wide spreads from higher-level abstraction for the human reader, logical behavior of control and data flow, into lower-level abstraction of machine language, unfortunately novice students have to deal with all this complexities in once.

Some visualization tools reveal the inner working of formal algorithm that canned in the system such as DataStructure Navigator [14] and Swan [15]. User may also develop their own algorithm rather than viewing limited predefined sample from the system, e.g. Raptor [16] , ANIMAL [17], and FLINT [18]. Other researchers put some efforts to reveal explicitly the process of the program through the dynamic artifacts called program or code visualization such as the phenomenal Jeliot [19] . A number of PVs also introduced some breakthrough concepts in visualizing the program behavior such as Ville [20] that proposed a language-independent feature to reduce the complexity of the diversity syntaxes of each programming languages. Moreover ALVIS Live! [21] Gave attention in immediate visual semantic and textual feedback.

FLINT system allows user to break down the problem in to sub-problems, the top-down decomposition forces user think sequentially as the nature of waterfall model. User constructs sub-solution using provided visual flowchart components. Once the solution has been generated, user is able to test their algorithm flows and changes in data value. This model of visualization has been enhanced by RAPTOR that enables user to develop his/her code incrementally rather in top-down sequence. It also provides built-in sub-programs with auto-completion to function names where it may be very useful for the expert user but likely would obscure the understanding of novice user.

Jeliot 3 is program visualization that interprets each line of the program into step by step execution in graphical symbols. It is intended to help novice programmers in understanding the behavior of the program. The separation between compile and execute environment caused the execution of the program into the dynamic animation of program behavior will be occurred only if the completed structure of the code is syntaxes error-free

Figure 1 . The main view of 3De-Synergetic PV



III. 3DE – SYNERGETIC PV

3De is an integrated algorithm-program visualization tool that can help students to construct their own algorithm, observe its logical behaviour and to get its auto-generated syntaxes. Even though it main purposes to help student in understanding the program process, instructors can use it to visualize their own programming or algorithm examples in lectures.

A. Key Features

The development of this visualization based on the integrated multi-representation model that proposed to help novice students to comprehend the multi forms of program from a higher level to a lower-level within integrated environment [22]. The basic idea of the model is to show novices about the programming stages starting from designing problem-solving, developing code and validating logical flow of the program through visualizing multi level of program abstraction. In this section, we present 3De's key features in three categories: programming related task, multi level of abstraction, and immediate semantic and syntax feedback.

1) Support on programming related tasks

One of the most important aspects of 3De is the ability to support three main tasks in programming simultaneously i.e. design, develop, and debug. User interacts with the system by constructing his or her own problem-solving design using provided elements that is called *plan.* System immediately will simulate each *plan* to explain to the user how this *plan* works and what the effect to the data values and then auto-generate syntax will display the proper syntax for corresponding *plan.* Parallel's views allow user to verify the produced output and the expected output in the same time while user in the developing process.

2) Display multi level of program abstractions.

The execution of designed algorithm can be viewed simultaneously in different level of abstraction; user can see it how the execution progresses from higher-level abstraction into dynamic explanation through intermediate-level of abstraction and finally the syntax translation of lower-level abstraction. Problem-solving design is constructed in the form of flowchart-like notation to ease the user in tracing the logical flow of their own design. Meanwhile system will simulate it using a sequence of highlighted dynamic text that will appear in program behaviour area, data monitor area and output area. Moreover, the converted *plan* will represented as textual syntax based on simple C++ form in the code area.

3) Provide immediate semantic feedback.

System can directly reveal the logical flow of the program, the value changing on the specific variable, and the output or display result of the program for each *plan* in which user applied as a part of his or her solution design. It means user does not have to wait to build all complete structure of problem-solving design to get visual apparent of the solution. Sequentially system will highlight each artefact that need to be focused by student to understanding how the process works, where the data come from or save into , the results and the textual code of the program. This key feature was motivated by the issues that delayed respond time and separation environment between run-time and edit-time causes the increment of internal working memory load and possibly can lead students lost their logical flow tracing and linking among level of abstraction.

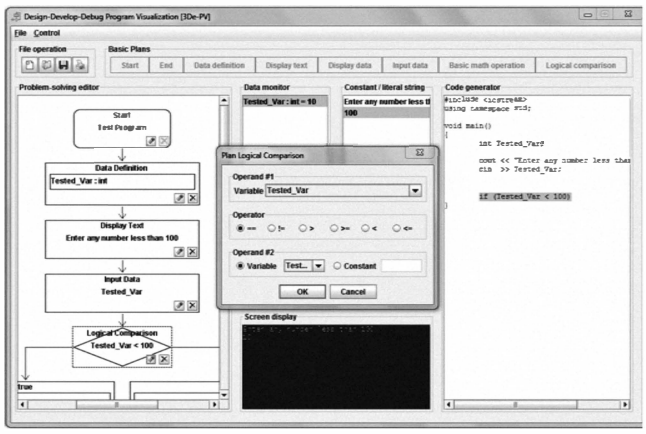


Figure 2. Constructing a problem-solving design using 3De

B. User interaction on 3De Synergetic PV

3De user interface built simply in one main form, which is, consists of five main sections: standard tool button, *plan* toolbar, problem-solving editor, program behaviour arena, and source-code area. There are two main interactions associated with the user of the application to the learning process, constructing and tracing

1) Constructing the problem-solving design

For constructing purposes, users develop their own problem solving design by compiling each *plan* in the problem-solving editor. To achieve their expected result of each *plan* user have to set the *plan*'s parameter, for example when user take the data definition *plan* they also have to define the real data for this *plan* through the parameter button within its *plan*. Parameter dialog for data definition will appear and user can sets the variable name, data type and initial value for his or her problem-solving requirement. System will display the *plan* with the parameter that has been set in the form of dynamic textual that allow user to observe and verify his or her own design

2) Tracing execution

For tracing purpose, user can browses each *plan* either forward or backward from constructed problem-solving design. System will re-executed all the *plan* starting from the START *plan* until selected *plan*, whereas the intermediate and lower level abstraction will display the current state of the selected *plan.*

IV. DISCUSSION

There are mutual and complex dependencies between understanding the conceptual framework of algorithm and the ability to construct and to debug a program. The ability to trace a program becomes one of the factors that are related to the ability to solve problems, and furthermore the ability of construct a problem-solving contributes to the programming skill. 3De attempts to support those comprehensive processes in understanding software development by giving the equal attention on design, development and debug program through visualization tool. Mostly available tools support only a part of programming-phase rather than continuing support through the programming-cycle. The lack of cross-referencing feature forces student to use different tool (e.g. data visualizer, algorithm visualization, program visualization, or debugger) for each stage to assist them in understanding and practicing skills in programming.

From the learner's point of view, 3De shows clearly how the interwoven relationship between algorithm design, the logical and data flow of program behavior, the translation into the syntax's program, and the output or result of the program. It has a potential to help student in understanding the essential behind basic programming concepts. We see an opportunity if each *plan* that added into a problem-solving design be presented both in lower level and higher level perspectives in concurrent way then it will maintain student's cognitive model quickly and leads to

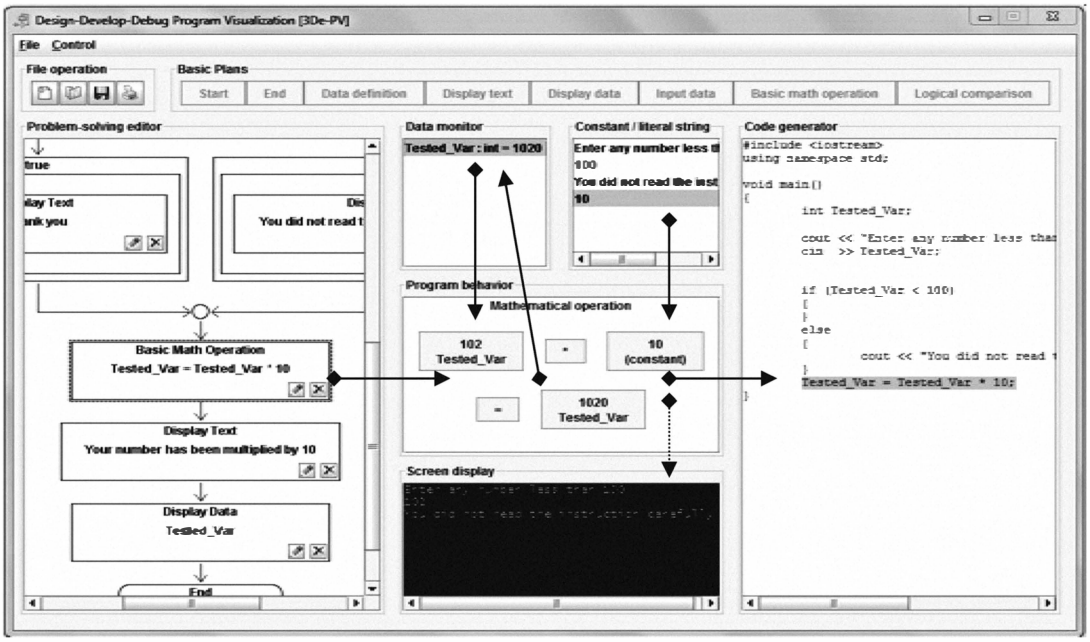


Figure 3. Sequence of abstract transformation from higher-level to lower-level

High-level understanding progressively. As well as the integrated environment with immediate semantic feedback that possibly can help student to reduce time needed to do edit-compile-run tasks repeatedly and increase the confidence feeling to the correctness of the program.

TABLE 1 COMPARISON BETWEEN 3DE, JELIOT3 AND RAPTOR

	3De	Jeliot3	Raptor
Object to be visualized			
Algorithm	yes	no	yes
Source Code	yes	yes	no
Data	yes	yes	yes
Level of abstraction			
High-level	yes	no	yes
Intermediate-level	yes	yes	no
Low-level	yes	yes	no
Supported task			
Design	yes	no	yes
Develop	yes	yes	no
Debug	yes	yes	yes
Semantic feedback			
On-request	no	yes	yes
Immediate	yes	no	no
Step forward	yes	yes	yes
Step backward	yes	no	no
Continuous running	no	yes	yes

However, there is some consideration that it is important for the learning process that student do programming by themselves in term of thinking, constructing (writing) and debugging in whether higher or lower levels of abstraction rather than rely on the abundant tools' features to do the thinking and auto constructing [23]. Regarding to the level of engagement taxonomy proposed by Naps et. al. [24], 3De's features covers all the engagement level from viewing, responding, changing, constructing and presenting.

There are some similar tools to 3De with some of common features, however; some striking differences provide different representation in level of abstraction among of them. Short comparison of similar tools describe in Table 1.

V. CONCLUSION

Learning to program is a hard process of trial and error, a challenging process that requires both cognitive and skill performance that involve a number of activities and some different forms of program representation rather than just learning to write the code or syntax in particular language. With this comprehensive process of design, develop, and debug and multi-level representation of program abstract within integrated edit-run environment, 3De-synergetic PV as learning-aid tool is needed to shift the internal working memory load of students to provide more *"space"* to the essential knowledge of programming.

Developed system focused merely on the basic knowledge related to the development of algorithms design and code construction in a simple C++ programming language, algorithm design has not yet included in the

iterative handling, procedure and function, and advanced data type. In the future 3De is going to be extended to cover some of those issues. Moreover, it is also going to be evaluated on the first programming courses at University of Teknikal Malaysia Melaka and University of Dian Nuswantoro Indonesia. In addition to the learning performance the evaluation will focus on student ability to trace and write the code and the viability of 3De's features.

ACKNOWLEDGMENT

This work is done with support from the University of Teknikal Malaysia, Melaka via the short term grant projects PJP / 2010 / FTMK (12E) S720, in collaboration scholarship from the University of Dian Nuswantoro, Indonesia.

REFERENCES

[1] B. Du Boulay, "Some difficulties of learning to program," *Journal of Educational Computing Research*, vol. 2, no. 1, pp. 57–73, 1986.

[2] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A meta-study of algorithm visualization effectiveness," *Journal of Visual Languages & Computing*, vol. 13, no. 3, pp. 259–290, 2002.

[3] P. Ihantola, V. Karavirta, A. Korhonen, and J. Nikander, "Taxonomy of effortless creation of algorithm visualizations," in *ICER 2005 - International Workshop on Computing Education Research*, 2005, pp. 123-133.

[4] V. Karavirta, A. Korhonen, L. Malmi, and K. Stalnacke, "MatrixPro - A tool for on-the-fly demonstration of data structures and algorithms," *In Proceedings of the Third Program Visualization Workshop*, 2004.

[5] R. B. B. Levy and M. Ben-Ari, "We work so hard and they don't use it: acceptance of software tools by teachers," *ACM SIGCSE Bulletin*, vol. 39, no. 3, p. 250, 2007.

[6] S. P. Reiss, "The Paradox of Software Visualization," *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pp. 1-5, 2005.

[7] T. L. Naps et al., "Exploring the role of visualization and engagement in computer science education," *ACM SIGCSE Bulletin*, vol. 35, no. 2, p. 131, Jun. 2003.

[8] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003.

[9] A. V. Maryhauser and A. M. Vans, *Program Understanding - A Survey*. 1994.

[10] R. Lister et al., "A multi-national study of reading and tracing skills in novice programmers," *ACM SIGCSE Bulletin*, vol. 36, no. 4, p. 119, Dec. 2004.

[11] B. Hanks, L. Murphy, B. Simon, and R. McCauley, "CS1 students speak: advice for students by students," *ACM SIGCSE*, pp. 19-23, 2009.

[12] K. Ala-mutka, *PROBLEMS IN LEARNING AND TEACHING PROGRAMMING - a literature study for developing visualizations in the Codewitz-Minerva project*. 2003, pp. 1-13.

[13] M. McCracken et al., "A multi-national , multi-institutional study of assessment of programming skills of first-year CS students A framework for first-year learning objectives," *ACM SIGCSE Bulletin*, vol. 33, no. 4, pp. 125-180, 2001.

[14] Jens-Peter Dittrich, "DSN: Data Structure Navigator," 2001. [Online]. Available: http://dbs.mathematik.uni-marburg.de/research/projects/dsn/. [Accessed: 25-Nov-2010].

[15] C. a Shaffer, L. S. Heath, and J. Yang, "Using the Swan data structure visualization system for computer science education," *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education - SIGCSE '96*, vol. 28, no. 1, pp. 140-144, 1996.

[16] M. C. Carlisle, T. A. Wilson, J. W. Humphries, and S. M. Hadfield, "RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving," *ACM SIGCSE Bulletin*, vol. 37, no. 1, p. 176, Feb. 2005.

[17] G. Robling, M. Schuler, and B. Freisleben, "The ANIMAL Algorithm Animation Tool," in *Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, 2000, pp. 37-40.

[18] U. Ziegler and T. Crews, "The Flowchart Interpreter for Introductory Programming Courses," *Proceeding of FIE '98 Conference*, pp. 307-312, 1998.

[19] A. Moreno, N. Myller, and R. Bednarik, "Jeliot 3, an extensible tool for program visualization," p. 183.

[20] T. Rajala, M. J. Laakso, E. Kaila, and T. Salakoski, "VILLE–A language-independent program visualization tool," in *The Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007),*, 2007, vol. 88, pp. 15–18.

[21] C. Hundhausen and J. Brown, "What You See Is What You Code: A 'live' algorithm development and visualization environment for novice learners," *Journal of Visual Languages & Computing*, vol. 18, no. 1, pp. 22-47, 2007.

[22] Affandy, N. S. Herman, S. Salam, and E. Noersasongko, "A Study of Tracing and Writing Performance of Novice Students in Introductory Programming," in *Software Engineering and Computer Systems*, 2011, pp. 557-570.

[23] S. James, M. Bidgoli, and J. Hansen, "Why Sally and Joey can't debug: next generation tools and the perils they pose," *Journal of Computing Sciences in Colleges*, vol. 24, no. 1, pp. 27–35, 2008.

[24] T. L. Naps et al., "Evaluating the educational impact of visualization," *Working group reports from ITiCSE on Innovation and technology in computer science education - ITiCSE-WGR '03*, pp. 124-136, 2003.