Transforming Source Code Examples into Programming Tutorials

Roger Rudolph Gajraj
Department of Computing & Information Technology
The University of the West Indies
St. Augustine, Trinidad
roger.gajraj@uog.edu.gy

Malcolm Williams
Department of Computer Science
The University of Guyana
Turkeyen, Guyana
malcolm.williams@uog.edu.gy

Margaret Bernard
Department of Computing & Information Technology
The University of the West Indies
St. Augustine, Trinidad
margaret.bernard@sta.uwi.edu

Lenandlar Singh
Department of Computer Science
The University of Guyana
Turkeyen, Guyana
lenandlar.singh@uog.edu.gy

Abstract—**One popular approach to teaching computer programming is to use example programs to demonstrate programming concepts. We propose to increase the pedagogical value of example program source code by transforming them into self-explaining tutorials within a learning integrated development environment. In this paper, we present a stepwise instructed implementation of annotated example code. Source code with instructor comments is parsed and processed to create an intelligent learner environment. Students are guided step by step to develop the program solution. Explanations are auto-generated for each line of code; these come from an author's comments as well as extended explanations dynamically generated for certain coding constructs. Explanations are presented to the learner in multiple modes using the full range of multimedia displays. Source code examples can be used as self-contained tutorials.**

Keywords-e-learning; educational technologies; programming pedagogy; source code examples; integrated development environment

I. INTRODUCTION

Teaching by example is a well established method for teaching computer programming. Experienced teachers of programming prepare suitable examples of program code which illustrate the particular programming construct that is being taught; students learn from these examples by seeing how the construct is used in problem solving as well as familiarizing themselves with syntax and semantics of the language. These example programs may be made available to the student in digital form and students may be encouraged to run the code in some IDE. Many programming textbooks use this approach of teaching by example.

In this paper, we present a computer aided example-based approach to teaching and learning programming. Students are guided step-by-step to construct the examples that the instructor has already prepared. We also present a learner's integrated development environment (L-IDE), called CSmart, which facilitates this guided instruction. The instructor creates annotated example programs which can illustrate the use of a programming concept. When the student selects that example problem, he is not shown the solution code immediately; rather he is guided through step-by step instructions so that he can develop the program solution which the instructor/expert has prepared. This is done dynamically, in real time, where the example source code with instructor's annotation is parsed and used to generate explanations of the programming construct in focus. The explanations generated are a combination of text and visual representations. The intelligence is built into CSmart so that instruction is not static but relates to the point in the program that the student is working on. CSmart stores values of variables in the program and is able to present to the learner exactly what is happening with the code that he is typing in.

This example-based pedagogical approach for teaching and learning programming actually combines the traditional approach of learning by example with learning by doing and learning using visualization. It can be used for novice programmers where the instructor examples are simple one-task programs, with possibly just one construct (a FOR loop, for example). The approach can also be used to develop more experienced programmers where the instructor examples focus not so much on syntax but on expert techniques for problem solving. Most instructors have a bank of example programs that they reuse with different student groups. The CSmart environment allows the instructor to create a repository of annotated examples, categorized and sequenced in any manner that the instructor prefers.

In the following sections, we present our pedagogical approach after highlighting related work and discussing their approaches. The CSmart environment is then detailed to show how it embodies our novel pedagogical concept by processing source code examples as tutorials themselves.

II. Background

A. *Related Literature*

In a study of difficulties faced by novice programmers, both students and teachers valued example programs as the most useful type of learning material [1]. Most sources of static examples are accompanied by explanations in varying degrees of detail and usefulness. Programmers conventionally seek example code in resource repositories such as tutorial web-pages, forums and books. These examples can then be implemented in a separate integrated development environment (IDE) and then compiled to run the resulting executable program. This is done to gain deeper understanding of what example code really does at runtime execution.

Using source code to aid in the learning of programming has been explored by environments such as 'WebEx' [2], 'Jeliot' [3] and 'BlueJ' [4]. 'WebEx' presents example source code with annotations explaining relevant lines of code. A read-only exploration of an example is done by altering the visibility of annotations. 'Jeliot' focuses on auto-generating visualizations of source code's execution at run-time. 'BlueJ' also parses source code like 'Jeliot' but to produce visualizations promoting object oriented concepts only. In addition, 'BlueJ' is a simple IDE that provides graphical and textual editing of source code which can be further compiled.

Of the aforementioned environments, only 'WebEx' focuses on directly leveraging pedagogical value from example source code by presenting textual explanations of individual lines of code. However, 'WebEx' hardly differs from reading static commented source code because it only allows interactivity with reading annotations which is similar to reading extended comments. Also, there is no IDE support for a user to gain insight from the experience of an example program's behavior at run-time.

None of the environments mentioned provide guidance for the actual activity of programming implementation. In addition to an instructor explaining example code in traditional lab environments, students are often guided to implement the example and encouraged to change and apply the example in order to solve a similar problem. Guided implementation of example code could assist in reducing cognitive overheads while learning to program.

B. *Pedagogical Approach*

We propose to increase the pedagogical value of source code by transforming them into self-explaining tutorials within an integrated development environment. This pedagogical approach to teaching and learning programming actually combines the traditional approach of learning by example with learning by doing and learning using visualization. This 3-pronged approach combines strategies which individually are well established in the literature as being effective for teaching and learning programming [1, 5].

Each line in a source code example can be presented as an instruction for a learner to actually type into the environment. We posit that this instructed implementation activity promotes learning by doing. In addition, the guidance to implement working code may reduce cognitive overheads that may hinder the learning process.

Teachers and books often try to explain source code examples line-by-line. Comments belonging to selected lines of source code can store an author's explanation of its intended purpose. As a result, we propose to use comments to annotate lines of source code. These annotations can be positioned strategically within an IDE at the same time the instructed implementation activity is carried out by the learner.

Visualization is also used as an effective pedagogical technique [5]. Visualization of what certain code constructs are doing can be provided as graphical annotations alongside the textual annotations. The approach taken is different from algorithm visualization, which focuses on visualizing an entire algorithm. We rather focus on visually explaining a line of code by itself. Visuals are presented as metaphorical representations of code semantics.

We call this entire pedagogical approach "stepwise instructed implementation of annotated example code".

III. Learner Integrated Development Environment

Traditional IDEs provide little support for novice programmers who attempt to learn programming from example code. Beginners may have to type example code into an IDE either from book sources or "copy and paste" from electronic media sources. We propose a "learner's integrated development environment" (L-IDE) concept which supports the use of example code to aid in the learning process.

We have created a L-IDE model which parses actual source code files into a presentable format more amenable to learning. A beginner is guided through implementation of an example program. Code comments are extracted and provided as annotations for respective lines of code. Visualizations are generated dynamically from code which attempt to give a graphic representation to aid understanding of abstract programming concepts. After guided implementation of example code, a beginner programmer has the option to edit the example and/or compile and execute the code with possible error feedback.

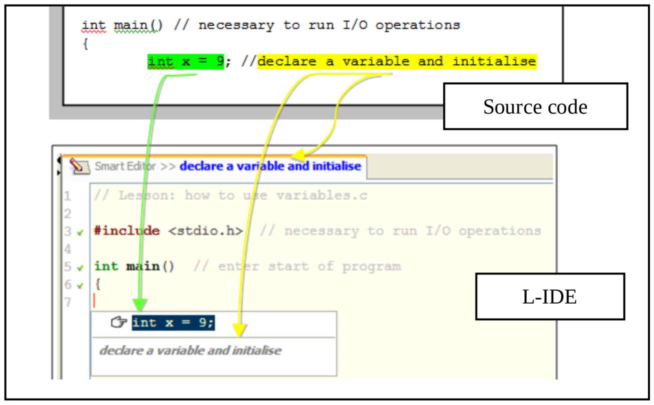
The following sub-sections describe the features of our L-IDE called 'CSmart'. Currently, CSmart is limited to parsing source code of the C programming language.

A. *Information Extraction*

Tutorials are created dynamically where example source code files are parsed and presented in a richer format that aids learning. Figure 1 shows how an actual line of code and relating comment is parsed from a source code file and displayed to the learner as an informal instruction within CSmart's editor.

B. *Stepwise Guided Implementation*

In applying the step-wise guided technique, the learner is instructed to type out each line of code in an example. This stepwise approach fosters learning through learning by doing. In the 'CSmart' L-IDE, the example code that is to be typed out, as part of encoding the program, is placed under the cursor – within a tooltip – in full view of the programmer, while they type the code on the line above the tooltip guide (see Figure 1 and Figure 2).



```
int main() // necessary to run I/O operations
{
        int x = 9; //declare a variable and initialise
```

```
// Lesson: how to use variables.c

#include <stdio.h>  // necessary to run I/O operations

int main()  // enter start of program
{

  int x = 9;
  declare a variable and initialise
```

Figure 1. Code and comment extraction from source code

Similarly, the comments that explain a line of code are placed just beneath the line of code. This allows very little overhead of copying code because it is very close to the input area, specifically, where the cursor is located within the editor area. We posit that the strategic presentation all of these pieces of information (i.e. presenting a lot of information in a small space without overloading the user) increases the information content of the user interface, and facilitates learning.

C. *Explanation of Code*

Example source code demonstrates select programming concepts which can be explored by the learner in order to foster learning and understanding. A line by line explanatory approach was used to explain example code. This allows learners to focus on the semantics of the programming language. To best explain examples visually and textually, Clark & Mayer provide the following tested guidelines in [6] which were considered and incorporated into the design:

- "multimedia principle" which involves appropriate use of text, audio and graphics
- "contiguity principle" ensuring words align to graphics
- "segmenting principle" where small manageable chunks of data are presented at a time to the user

These principles were applied to explain lines of code visually, textually and audibly. These techniques are further illustrated in the proceeding sub-sections.

1) *Annotation from Comments*

Comments from source code files are used to provide auto-generated annotation of the corresponding lines of code that the learner types out in the L-IDE text editor. Where no comments exist in the example source code, no auto-generated explanation for those lines of code is generated. Authors of tutorials (example code) are encouraged to create better tutorials by including comments in each line of code. It is the human tutor's responsibility to ensure appropriateness of explanation of lines of code.

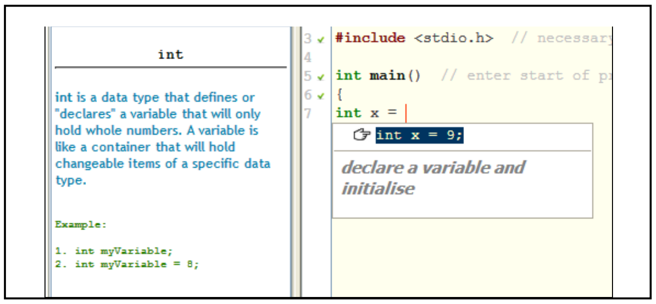
In addition, there is an automatic display of extended explanations of certain coding constructs. Certain coding constructs may be keywords or popularly used functions of the C programming language. These extended explanations intelligently appear next to the editor area whenever certain coding constructs are used in a line of code in order to augment the learning process.

For example, a line of code and comment within a source code file may be:
int x = 9; //declare a variable and initialise

From the previous line of code above, Figure 2 shows how the comment – as an explanation – is displayed just under the instructed line of code to be typed out.

2) *Information pane for Keywords and Functions*

The CSmart environment displays additional information when keywords and functions from the C standard library are found while parsing lines of code. This intends to give a learner a better understanding of how to use keywords and functions.



int

int is a data type that defines or "declares" a variable that will only hold whole numbers. A variable is like a container that will hold changeable items of a specific data type.

```
Example:

1. int myVariable;
2. int myVariable = 8;
```

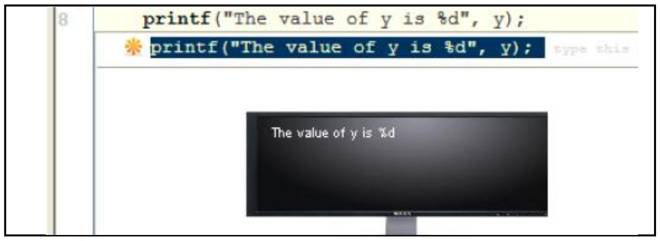
```
#include <stdio.h>  // necessary

int main()  // enter start of pr
{
int x = |
  int x = 9;
  declare a variable and
  initialise
```

Figure 2. Textual explanation of instructed line of code to be typed out

For example, the left pane in Figure 2 beside the editor dynamically displays additional information on the 'int' - integer - data type that is found in the line of code.

3) *Visual explanation of source code*

Visual explanations of code constructs are automatically generated just under comments that explain an instructed line of code that is being typed out. Visuals attempt to explain coding constructs with very little prerequisite of programming concepts. Beginners may understand programming concepts easier by being able to relate to visuals.



```
printf("The value of y is %d", y);
  printf("The value of y is %d", y);   type this
```

```
The value of y is %d
```

Figure 3. Visualization for output function 'printf'

For example, the popular 'printf' statement for output to screen is portrayed (see Figure 3) by showing the output on an image of a computer monitor. The intention is to get the learner to understand what printf does – it outputs text to screen. In another example (see Figure 4), variables are portrayed as containers because a container can hold an item like a variable holding data.

Calculation and assignment based visuals are dynamically generated from a trace-table data structure that keeps track of the data in variables. The trace table is used to show what data is moved and/or replaced within variables during calculations and assignments. An extension of the java programming language called 'Groovy' [7] was used to evaluate loop and conditional expressions and to provide the inputs into the visualization module. Figure 4 shows a calculation where the data in variable 'x' is added to the number seven (7) to give the sum of sixteen (16) that is further animated by moving the number sixteen (16) into the 'y' container representing the 'y' variable. This animated path is represented by the broken line in Figure 4.

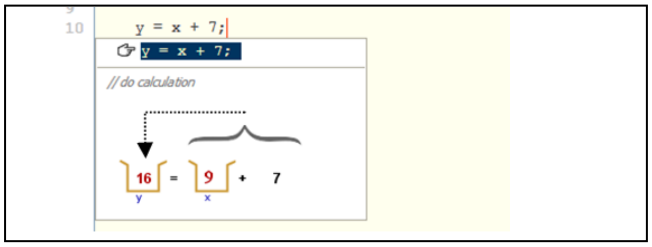


Figure 4. Visualization of calculation and assignment of data to variable

D. *Compilation and Execution*

The 'CSmart' L-IDE provides the opportunity to compile and execute implemented source code during a tutorial. Learners can see how a program behaves from its implemented example code.

Additionally, users may experiment with example code in terms of changing literals and variables in the original example. Compilation of modified code with error feedback is possible with additional support from an incorporated C compiler.

IV. Discussion

The learner integrated development environment's ability to create tutorials directly from source code infers a high ease of adoption by instructors as a teaching tool. There is neither a new method nor syntax to learn in order to create a tutorial. Tutorials are source code examples themselves and the quality would vary based on how instructors annotate lines of code with comments. Repositories could store high quality source code examples for re-use.

Instructors can focus on helping students individually instead of losing time to write and explain examples verbally. Students are able to go through tutorials at their own pace via a computer based pedagogical resource which could promote better learning.

As classroom sizes increase, students gain less attention from instructors. However, all students will be exposed to the same quality of pedagogy via a L-IDE such as 'CSmart'.

V. Conclusion and Future Work

This paper presents a work in progress, where a development environment is being used to deliver programming tutorials from source code examples. However, the pedagogical presentation of visual and textual explanations together with interactive implementation is constrained by the content within a source code example. Therefore, the quality and context of a tutorial is controlled by an example source code's content; code is instructed to be typed in an editor while it's relating comments are presented as annotations.

Experimental evaluation of the combined pedagogical approach, within the L-IDE, is the next step for future research. Initial evaluation of this pedagogical approach, within the 'CSmart' environment, was encouraging as reported in [8]. Participants in the evaluation rated their understanding of tutorials at a mode of eighty percent (80%). All participants reported that the L-IDE was helpful and they felt that they learnt some programming. These results were further supported by the following unsolicited comments from the participants:

- "made concept simple"
- "it helped to make things a little more understandable"
- "at least I learnt something by doing the tutorials"
- "boosted my knowledge"

Visualization could be enhanced by fading from a metaphorical representation of source code execution to a machine level representation in order for learners to gain a deeper understanding of how a program works at run-time.

If future evaluations show a positive effect of the pedagogical approach in a real learning process, students can use a L-IDE environment to learn programming on their own, just as if an instructor is there to guide them in person.

References

[1] E. Lahtinen, K. Ala-Mutka, and H. Järvinen, "A study of the difficulties of novice programmers," Proc. 10th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '05), ACM, 2005, pp. 14-18, doi:10.1145/1067445.1067453

[2] P. Brusilovsky, I. Hsiao, and M. Yudelson, "Annotated program examples as first class objects in an educational digital library," Proc. 8th ACM/IEEE-CS joint conference on Digital libraries (JCDL '08), ACM, 2008, pp. 337-340, doi:10.1145/1378889.1378946

[3] A. Moreno, N. Myller, M. Ben-Ari, and E. Sutinen, "Program animation in jeliot 3," Proc. 9th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '04). ACM, 2004, pp. 265-265. doi:10.1145/1007996.1008099

[4] M. Kolling, B. Quig, A. Patterson, and J. Rosenberg, "The BlueJ System and its Pedagogy," Journal of Computer Science Education, Special Issue on Learning and Teaching Object Technology, Vol 13, No 4, Dec 2003.

[5] J. Bergin, K. Brodie, and M. Patiño-Martínez, "An overview of visualization: its use and design: report of the working group in visualization," Proc. 1st conference on Integrating technology into computer science education (ITiCSE '96), ACM, 1996, pp. 192-200, doi:10.1145/237466.237647

[6] R. Clark and R. Mayer, “E-learning and the science of instruction,” 2nd ed., San Francisco: Pfeiffer, 2008.

[7] D. Koenig, A. Glover, P. King, G. Laforge, and J. Skeet, “Groovy in Action,” Manning Publications Co., Greenwich, CT, USA, 2007.

[8] R. Gajraj, “A Computer Based Programming Pedagogy: stepwise instructed implementation of explained example code,” Thesis, unpublished.