Integrated Learning Development Environment for Learning and Teaching C/C++ Language to Novice Programmers

Sana'a M. Algaraibeh
Computer Science
University of Idaho
Moscow, ID, USA
alga0486@vandals.uidaho.edu

Tonia A. Dousay
Curriculum and Instruction
University of Idaho
Moscow, ID, USA
tonia@uidaho.edu

Clinton L. Jeffery
Computer Science
University of Idaho
Moscow, ID, USA
jefferyc@uidaho.edu

Abstract-**This Innovative Practice Work-in-Progress paper presents an Integrated Learning Development Environment (ILDE) that integrates technologies with pedagogies for first-year students learning to program. Novice programmers must overcome misconceptions, debugging, and problem-solving. ILDE employs multimedia learning content, formative feedback, a customized compiler, and visualization using modern pedagogical and cognitive psychology practices. Visualization and multimedia illustrate what happens inside the computer as the program is running. Enhanced compiler messages with graphical representation reduce the difficulty of compilation errors.**

Keywords—CS education, learning and teaching, C/C++ language, IDE, novice programmers, programming introductory course

I. Introduction

A student in the first year computer science course finds themselves like Alice in Wonderland. There are many new mysterious concepts and they must acquire a large amount of knowledge and many new skills. Programming is a primary competency and a prerequisite for almost all CS courses. It is the basic building block in their journey. Many CS educators express that their students lack programming skills, even after prior programming courses [1]. Much like Wonderland, learning how to program feels like a multidimensional journey filled with different perspectives and tasks.

This paper presents an Integrated Learning Development Environment (ILDE) to enhance the learning and teaching of programming for first- and second-year computer science students. ILDE integrates technologies with pedagogies. The design of the ILDE model is built upon Cognitive Load Theory [2],[3], Kolb's Experiential Learning Theory [4], Constructivism principles [5], Cognitive Theory of Multimedia Learning [6], software visualization technology, and an educationally customized compiler. The learning content addresses core programming competencies taught in computer science education for first and second year post-secondary university students. ILDE merges related course topics: Computational Thinking and Problem Solving, Programming Languages, Computer Operating Systems, System Software, and Discrete Mathematics. Further, ILDE uses problems and projects from real-life contexts to support meaningful learning. The learning activities target complex programming skills developed from related subskills.

The ILDE model is based upon two hypotheses: 1) ILDE's Multimedia Learning Content deploys visualization of computation to enhance novices' performance and 2) ILDE's Informative Feedback that utilizes a customized compiler will enhance novices' performance. For both hypotheses, novice performance will be measured by observing whether using ILDE changes course success rates (defined as a grade of C or above) in a statistically measurable manner.

II. Challenges Facing Novice Programmers

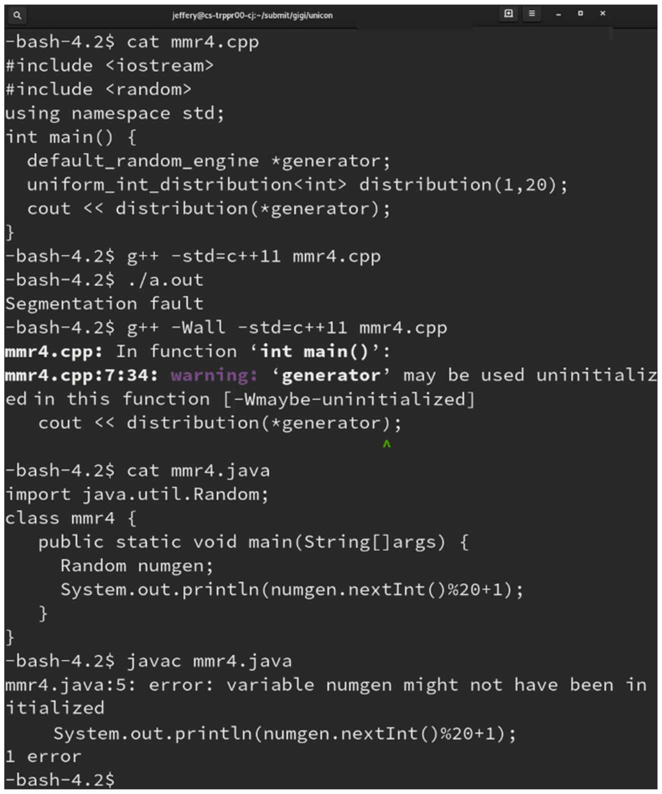
Learning to program is difficult. Required competencies include comprehension of programming concepts and ability to write code. Students must learn to convert real-world problems into computer solutions. Fuller et al.'s [7] learning taxonomy sums up the objectives of introductory programming courses as follows: recognize, understand, analyze and evaluate the programming concepts; apply these concepts by writing program code to solve a problem similar to the problems already learned; create program code to solve new problems. The learner must also understand the machine operations that a programming language expresses [8]. Why is it so difficult to teach or learn to program? Novices find programming challenging due to fundamental misconceptions, as well as their lack of debugging and problem-solving skills [8]-[10].

A. *Misconception*

Coding misconceptions arise from a lack of knowledge or a false assumption. For example, students may misunderstand the mechanism of a loop, or the relationship between language constructs and underlying memory usage [8, 10, 11, 12 , 13, 14]. ILDE has an innovative method for fixing misconceptions. Consider uninitialized memory allocation, a common error for novices: it arises due to a misconception about the relation between memory and language elements. Fig. 1 shows C++ and Java code that uses an object without initialization. When compiling mmr4.cpp, g++ doesn't report an error; when it is run, a "Segmentation fault" message appears. This message is mysterious to novices and doesn't help them understand or fix their problem. Encouraging novices to take the compiler's warnings seriously can help. g++ -Wall reports " 'generator' may be used uninitialized in this function". The Java compiler produces a "variable number might not have been initialized" error. Microsoft Visual Studio and BlueJ IDE show similar error messages for this type of error.

To address this misconception, ILDE builds a user profile that tracks the user's progress and diagnoses their level. The learning content has information about the exercises, such as the goal and plan. ILDE utilizes the user profile on current and past exercises to give tailored compiler messages. For the "uninitialized object" misconception at beginner levels, the ILDE feedback subsystem plays a video that explains the memory allocation process and how uninitialized memory affects program logic. For learners at the intermediate level, the feedback subsystem invokes an image that shows the memory simulation of the error with a link to the related lessons.

However, programming concepts are like icebergs that hide a lot of details. On one hand, researchers of the psychology of programming urge the programming language designer to employ cognitive principles to lower the barriers for programming[15]. On the other hand, it is very important that the learner understands the 'inner world' of the programming language being used [8,15]. Hoc and Nguyen-Xuan added that programming acquisition is about learning basic operating rules of the processing device that underlies the language; the constraints of these operations upon the program structure; the relation between task structure and programming language semantics [16]. So, understanding the smaller instructions that construct the programming statements and connecting these instructions to the concepts of memory and CPU operations may solve the issue of the fundamental misconceptions.



```
-bash-4.2$ cat mmr4.cpp
#include <iostream>
#include <random>
using namespace std;
int main() {
  default_random_engine *generator;
  uniform_int_distribution<int> distribution(1,20);
  cout << distribution(*generator);
}
-bash-4.2$ g++ -std=c++11 mmr4.cpp
-bash-4.2$ ./a.out
Segmentation fault
-bash-4.2$ g++ -Wall -std=c++11 mmr4.cpp
mmr4.cpp: In function ‘int main()’:
mmr4.cpp:7:34: warning: ‘generator’ may be used uninitializ
ed in this function [-Wmaybe-uninitialized]
   cout << distribution(*generator);
                                  ^

-bash-4.2$ cat mmr4.java
import java.util.Random;
class mmr4 {
   public static void main(String[]args) {
     Random numgen;
     System.out.println(numgen.nextInt()%20+1);
   }
}
-bash-4.2$ javac mmr4.java
mmr4.java:5: error: variable numgen might not have been in
itialized
    System.out.println(numgen.nextInt()%20+1);
1 error
-bash-4.2$
```

Fig 1: Example of misconception: "uninitialized memory allocation"

B. *Error handling and debugging*

Novices often fail to write the program structure correctly, misspell keywords, or omit or disorder the components of a program structure [11], [17]. Even modern compilers and IDEs frequently report inadequate, unclear error messages. Enhancing compiler error messages has a significant positive effects on novice programmers' learning by reducing the number of errors, and the number of repeated errors [18]-[20].

A logic error is when the produced program does not perform the intended functionality, for example due to improper converting between data types, or incorrect division of a float number by an integer [11], [17]. Ettles, Luxton-Reilly, and Denny report that misconceptions are the source of logic errors [12]. Finding and fixing logical errors is more difficult than fixing syntax errors. Debugging skills are difficult and even good programmers often lack these skills [12].

Many tools have been integrated into IDEs to enhance messages for both syntax and logic errors. Specialized IDEs have been created for education purposes. These efforts include PROUST[22], Merr [23], Expresso[11], BlueJ [24], TigerJython [13], Alice [25], Scratch [25], Greenfoot [25], WebTigerJython [26]. These projects are focused efforts targeting specialized aspects of programming instruction. Thus, novices still need assistance mastering basic concepts.

ILDE is designed to be used in the introductory programming courses where it focuses on the basic programming concepts and offers informative feedback. Other specialized educational IDEs prioritize support for the object-first approach that focuses on recognizing the object-oriented concepts: class, object, and members. In contrast, ILDE focuses on tailored feedback and learning of basic concepts: variable, data type, mathematical and logical operation, expression, iteration structure, if statement, and function. Other environments use block-based and graphic objects and are intended for younger learners. Those environments are designed to cover a small subset of programming concepts. Also, using graphic objects representing program constructs that are inserted and moved within a graphical environment is tricky, because of the high level of abstractions. Such visual programming environments are easier to use than text-based environments, but may make it harder to learn the underlying programming concepts[15, 16, 27].

III. ILDE

A. *Pedagogy*

ILDE draws upon Cognitive Load Theory, Kolb's Experiential Learning Theory, Constructivism principles, and Cognitive Theory of Multimedia Learning (CTML) [2]-[6]. The learning material is divided into phases that depend on learner performance levels. The programming concepts are highly-interactive elements. The sequence of the phases depends on the interactive relation between programming elements. Phase X is not accessible to the learner until she/he reaches the intermediate or better level of performance in the phases that phase X depends on. The levels of the learner are beginner, elementary, intermediate, advanced, and expert. The learning content introduces one programming concept at a time. The first lessons in each phase are easy, with direct instructions. The lessons present worked examples, followed by guided practices and simple and detailed feedback messages to lead the learner to develop the skills and achieve fluency. Then ILDE presents the complex and related concepts together with higher-order reasoning and thinking problems. ILDE's feedback subsystem provides support and scaffolding. The sequence of instructions and lessons in each phase follows Kolb's learning cycle.

The design of multimedia learning content follows the assumptions of CTML, such as the dual channels assumption, where ILDE employs verbal and visual learning content, and the active processing assumption, which entails that ILDE's learning content is a collection of self-paced training materials. Also, the same content is delivered in different ways. Formative feedback in the interactive learning environment has major potential to enhance the learning process and it can be effective in a well-defined domain [28].

B. *Multimedia Learning Content with Visualization Tool.*

The MLC presents programming concepts using video, audio, and text to explain programming structure, meaning, usage, and behavior. MLC is integrated with a visualization tool that displays what is happening inside the computer at compilation and runtime. Moreover, ILDE has a mascot named Reynold, a squirrel character helper who presents the content and guides learners through the environment.

The following scenario illustrates how the ILDE works. The learner opens the learning content menu. The first lesson is a video showing the course project. The objectives of the first lesson are: to introduce the potential of programming to the learners and to let students recognize what they will be able to do at the end of the course. The project is keeping financial records of a poultry production farm, an inventory management problem. The lesson starts with a farmer asking the students to help him to keep the financial records of his business. Then the video presents the software requirements of the project. That is "the software must keep records of receipts, expenses, and purchases of egg production and aggregate them by day, week, month, and year". Then the lesson will show what a solution for the project looks like and how the farmer is using it.

The second lesson starts by playing a video showing students how to develop their first program. It is similar to the well-known "Hello world" program, but they will write the name of the poultry farm. The video shows them how to use ILDE to write the program, to compile it, to correct errors, and to run the program. In the second activity of the lesson, students will write the program by themselves. Reynold will guide them, by pointing where to write, what button to press to compile, and so on. The task text will be available as an image dockable to the edge of the screen while they write their program. This lesson is the first stage of the learning cycle of Kolb's experiential learning style theory. It puts the learners in a concrete experiment. Then it is followed by a reflective activity; the second stage of the learning cycle. Students fill a form of questions asking them about what they see, and how they do the task. The third activity is an explanation of how computers are dealing with their program. It is a video that shows the students a visualization of the fetch-execute cycle. It is a simple explanation of how the memory and CPU work in their first program. It shows the phases that the program passes through, from source code to an executable program.

The lessons will gradually introduce programming concepts. Following cognitive load theory recommendations, each lesson focuses on one concept at a time. In addition, it takes into consideration the different skills that must be developed, followed by exercises with no context to master these skills.

A lesson later explains the C language for control structure, a main concept of programming. The lesson starts with a video explaining the iteration structure in the C language. It shows the anatomy of the for control structure. For example, the anatomy of the statement

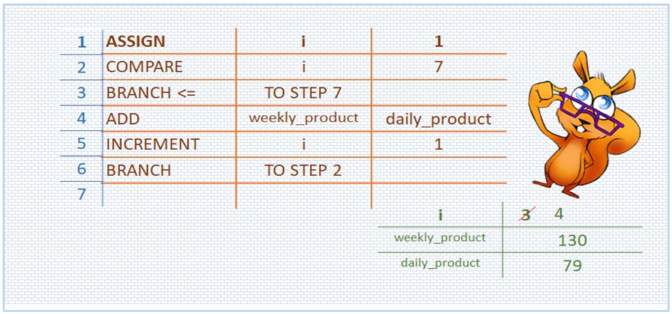
```
for(i=1; i <=7; i++) weekly_product +=  daily_product  ;
```

can be illustrated by:

```
 1:Assign: i=1;
2: Compare: i, 7;
3: Branch (go to) if less than:  step 7.
4: ADD: add the daily product to weekly product;
5: Increment: add 1 to the counter i;
6: Branch (go to) to the checkpoint of the loop step 2;
7: Rest of the code.
```

The lesson explains the usage of the iteration control structure and connects it with a simple problem from the course project. The learners will compute the production for a week. To repeat the entries for each day of a week, they will need to use the iteration structure. MLC introduces each programming concept with a real-world context. This helps the learners to scaffold their knowledge and retain it easier.

The lesson explains a language structure behavior using visualization tools. In the next activity, the lesson opens two screens. One shows a sample program's source code and the other shows the visualization of the program. An audio explanation accompanies the visualization screen, which shows how the computer interprets a language structure via a series of smaller instructions, including the variables used in a memory simulation picture and the flow of control. The visualization window shows the fetch-execute cycle for this example. It shows how the CPU works in the program and how the values of variables in memory change accordingly after each cycle. Fig. 2 and Fig. 3 show the visualization of memory and CPU consequently. After reviewing the example's source code execution, the learner can experiment with the source code and the visualization window reflects the change.



1	ASSIGN	i	1
2	COMPARE	i	7
3	BRANCH <=	TO STEP 7	
4	ADD	weekly_product	daily_product
5	INCREMENT	i	1
6	BRANCH	TO STEP 2	
7			

i	~~3~~ 4
weekly_product	130
daily_product	79

Fig. 2 Example of memory visualization of " for(i=1; i<=7; i++) weekly_product +=daily product; "

C. *Informative Feedback from Customized Compiler*

The feedback subsystem is integrated with a customized compiler that offers tailored error messages. The MLC has exercises guided by an e-booklet. As the learner works on solving lesson problems, they may consult this e-booklet with step-by-step instructions delivered by Reynold, who highlights the relevant part of the code. Once the learner writes the source code in the editor and compiles it, the ILDE will tailor informative feedback related to the error, problem, and level of the lesson. The ILDE will also display frequent and common errors in an interactive graphical representation.

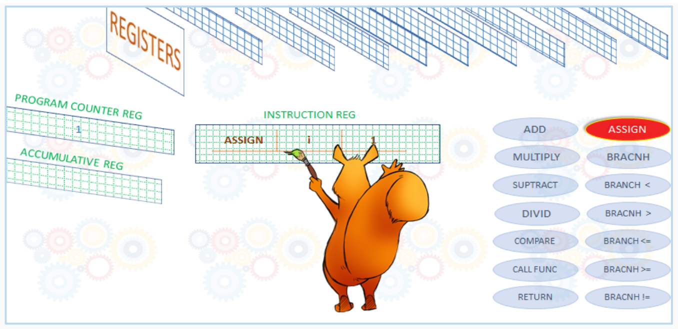


Fig. 3 Example of CPU visualization

The e-booklet learns the student's problem-solving skills. As it graduates from easier to more complex problems, it follows a problem-solving methodology: it teaches students how to extract the requirement and what the results should be; how to design a solution; how to implement it; how to test it. The e-booklet is an interactive screen that allows the user to write the requirements in a textbox, and the design of the problem in a textbox, then write the code in the editor. The students can compare their writing of requirements and design with the correct answers. While in the editor, the customized compiler produces the error messages. Reynold talks and explains the problem and points to relevant parts of the screen.

The customized compiler takes the learner level and the problem the learner works on, and gives tailored informative messages about the syntax errors. The feedback subsystem opens the related lesson video for common reported errors.

At the first level, students need intensive feedback while they work on tiny programs. Knowing the problem and the expected results enable the feedback subsystem to give precise tailored messages, and also connect these messages with the related lessons. The customized compiler compares the part of the student's code which is located before the detected error with possible correct solutions. This allows it to predict what statement the student is trying to write and explain what the error is in it.

The help menu has pictures of the programming statements syntax and function signatures. It enables the learner to dock these pictures on the edges of the working area of ILDE. Clicking on a syntax picture copies the content as text that can be pasted, adding a statement into the editor. Moreover, the feedback subsystem uses comic pictures accompanied by sounds when pointing to the errors. For example, common errors novices frequently fall into are missing semicolons, or missing braces. When this occurs, an animated icon with music sounds as the code is corrected. This will make an unstressed and memorable learning experience.

IV. Evaluation

We will evaluate the ILDE model by experiments during learning to assess the effects of the ILDE's feedback and learning subsystems on students' performance. We will design and implement two different experiments.

The first experiment will test the first hypothesis : " ILDE's Multimedia Learning Content can deploy visualization of computation to enhance novices' performance". The setting of the experiment will involve enabling ILDE's MLC features and disabling the ILDE's feedback subsystem features and compare the effects of this situation with regular IDE, on novice performances' acquisition of skills.

The second experiment will test the second hypothesis: "ILDE's Informative Feedback that utilizes a customized compiler will enhance novices' performance". The setting of the experiment will involve enabling ILDE's feedback subsystem features and disabling the ILDE's MLC feature and compare the effects of this situation with regular IDE on novice performances' acquisition of skills.

Both experiments will follow a quasi-experimental, pre-test/post-test design. Participants of the experiments will be undergraduate students of computer programming classes at the computer science department in a public research university.

V. Summary

ILDE is a specialized environment for learning and teaching novice programmers. The innovative aspects of ILDE are:

- The integration of learning content into the environment using modern pedagogical and cognitive psychology practice.
- Visualization of memory allocation and CPU operations. That makes it possible to explain the hidden parts of programming and visualize dynamically what happens inside the program.
- Feedback subsystem with customized compiler. Integration of learning content with ILDE enables the feedback subsystem to know the context of problems that students work on. This enables ILDE to give tailored error messages to the learner.

ILDE is designed for C/C++ which is difficult to learn and has a complex syntax. Many college CS curricula persist in beginning with these languages in their first courses for majors, and with good reason. But their relative difficulty makes the need for an educational learning development environment greater, rather than lesser than is needed in "easier" first languages.

The next steps involve continuing to work on the proposed technical solutions of the customized compiler and MLC. Then, we will finish the implementation with plans to test ILDE next academic year, 2020-2021.

Finally, the visualization features of ILDE may be used to learn other topics, such as data structures, algorithms, and programming language design. Also, with selection of an appropriate subset of the curriculum, ILDE may be used effectively in K-12 computer education.

References

[1] "Working group reports from ITiCSE on innovation and technology in computer science education," ITiCSE-WGR '01, Canterbury, UK, 2001, pp.125–180, doi: 10.1145/572133.572137.

[2] J. Sweller, J. van Merriënboer, and F. Paas, "Cognitive architecture and instructional design: 20 years later," Educ Psychol Rev 31, pp. 261–292 , Jan. 2019, doi:10.1007/s10648-019-09465-5.

[3] A. J. Martin and P. Evans, "Load reduction instruction: Sequencing explicit instruction and guided discovery to enahnce students' motivatin, engagment, learning, and achivemnt," In Advances in Cognitive Load Theory, S. Tindall-Ford, S. Agostinho, and J. Sweller, London and New York, UK and US: Routledge, 2020. pp. 13-29.

[4] D. Kolb, "The process of experiential learning," In Experiential Learning: Experience As the Source of Learning and Development," Prentice Hall, 2014, ch. 2.

[5] C. Bonk, D. Cunningham, "Searching for learner-centered, constructivist, and sociocultural components of collaborative educational learning tools," In Electronic Collaborators: Learner-Centered Technologies for Literacy, C. Bonk, K. King Eds. New York, NY, USA: Routledge, 1998, ch. 2, pp. 25–50.

[6] R. Mayer, "Cognitive theory of multimedia learning," In The Cambridge Handbook of Multimedia Learning , R. Mayer Ed. Cambridge, UK: Cambridge University Press, 2014, ch. 3, pp. 43-71, doi:10.1017/CBO9781139547369.005.

[7] U. Fuller, et al. "Developing a computer science-specific learning taxonomy," ACM SIGCSE Bulletin, vol. 39, no. 4, 2007, pp. 152–170.

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

[9] A. F. Blackwell, "First steps in programming: A rationale for attention investment models," Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments, Arlington, VA, USA, 2002, pp. 2-10.

[10] J. Bonar, and E. M. Soloway, "Pre-programming knowledge: A major source of misconceptions in novice programmers," Human-Computer Interaction, vol. 1, 1985, pp. 133–162.

[11] M. Hristova, A. Misra, M. Rutter, and R. Mercuri. "Identifying and correcting Java programming errors for introductory computer science students," SIGCSE '03, ACM, New York, NY, USA, 2003, pp. 153–156, doi: 10.1145/611892.611956.

[12] A. Ettles, A. Luxton-Reilly, and P. Denny "Common logic errors made by novice programmers," ACE '18, ACM, New York, NY, USA, 2018, pp. 83–89. Doi:10.1145/3160489.3160493.

[13] T. Kohn, "Teaching Python programming to novices: Addressing misconceptions and creating a development environment," PhD Thesis, ETH Zurich, Zürich, 2017.

[14] L. Kaczmarczyk, E. Petrick, J. East, and G. Herman. "Identifying student misconceptions of programming". SIGCSE'10 ACM, Milwaukee, Wisconsin, USA, 2010, pp. 107-111, doi: 10.1145/1734263.1734299.

[15] T. Green, "Language design and acquisition of programming: Programming languages as information structures ," In Psychology of Programming, J. Hoc, T. Green, R. Samurcay, D. Gilmore Eds. , Academic Press, ch 2. Sec. 2, pp. 117-137.

[16] J. Hoc, and A. Nguyen-Xuan "Language design and acquisition of programming: Language semantics, mental models and analogy," In Psychology of Programming, J. Hoc, T. Green, R. Samurcay, D. Gilmore Eds. , Academic Press, ch 2. Sec. 3, pp. 139-156.

[17] A. Altadmri and N. Brown, "37 million compilations: Investigating novice programming mistakes in large-scale student data," SIGCSE '15, ACM, New York, NY, USA, 2015, pp. 522–527, doi:10.1145/2676723.2677258

[18] T. Kohn, "The error behind the message: Finding the cause of error messages in python," SIGCSE '19, ACM, New York, NY, USA, 2019, pp. 524–530, doi:10.1145/3287324.3287381.

[19] M. Nienaltowski, M. Pedroni, and B. Meyer, " Compiler error messages: What can help novices?," SIGCSE '08, ACM, New York, NY, USA, 2008, pp. 168–172, doi:10.1145/1352135.1352192.

[20] B. Becker, P. Denny, R. Pettit, D. Bouchard, D. Bouvier, B. Harrington, A. Kamil, A. Karkare, C. McDonald, P. Osera, J. Pearce, and J. Prather, "Compiler error messages considered unhelpful: The landscape of text-based programming error message research," ITiCSE-WGR '19, ACM, New York, NY, USA, 2019, pp. 177–210, doi:10.1145/3344429.3372508.

[21] M. Ahmadzadeh, D. Elliman, and C. Higgins, "An analysis of patterns of debugging among novice computer science students," ITiCSE '05, ACM, New York, NY, USA, 2005, pp. 84–88, doi:10.1145/1067445.1067472.

[22] W. L. Johnson and E. Soloway, "PROUST: Knowledge-based program understanding," in IEEE Transactions on Software Engineering, vol. SE-11, no. 3, pp. 267-275, March 1985.

[23] C. L. Jeffery, "Generating LR Syntax Error Messages from Examples." ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 25, no. 5, 2003, pp. 631–640.

[24] 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.

[25] S. Fincher, S. Cooper, M. Kölling, and J. Maloney, "Comparing alice, greenfoot & scratch," SIGCSE '10, ACM, New York, NY, USA,2010, pp. 192–193, doi:10.1145/1734263.1734327.

[26] N. Trachsler, "WebTigerJython - A Browser-based programming IDE for education," Master's Thesis, ETH Zurich, Zurich, 2018.

[27] T. Green and M. Petre, "Usability analysis of visual programming environments: A 'cognitive dimensions' framework", Visual Languages and Computing, pp. 131-174, 1996.

[28] I. Goldin, S. Narciss, P. Foltz, and M. Bauer, "New directions in formative feedback in interactive learning environments," International Journal of Artificial Intelligence in Education, vol. 27, no. 3, pp. 385–392, Sep. 2017, doi: 10.1007/s40593-016-0135-7.