


Evaluations of Integrated Programming Environment for First-Year Students in Computer Engineering


Matias Salinas

(Pragmatics Labs, Coquimbo, Chile)

 <https://orcid.org/0000-0002-9408-9418>, madasaso@gmail.com


Paul Leger

(Escuela de Ingeniería, Universidad Católica del Norte, Coquimbo, Chile)

 <https://orcid.org/0000-0003-0969-5139>, pleger@ucn.cl


Hiroaki Fukuda

(Shibaura Institute of Technology, Tokyo, Japan)

 <https://orcid.org/0000-0003-1228-3186>, hiroaki@shibaura-it.ac.jp


Nicolás Cardozo

(University of the Andes, Bogota, Colombia)

 <https://orcid.org/0000-0003-1565-4106>, n.cardozo@uniandes.edu.co


Vannessa Duarte

(Escuela de Ciencias Empresariales, Universidad Católica del Norte, Coquimbo, Chile)

 <https://orcid.org/0000-0001-5399-6620>, vannessa.duarte@ucn.cl

Ismael Figueroa

(Pragmatics Labs, Coquimbo, Chile)

 <https://orcid.org/0000-0003-3661-4963>, ifigueroap@gmail.com

Abstract: Many factors influence the problems that currently exist in the learning-teaching process of programming. The use of an Integrated Development Environment (IDE) makes the experience a complicated process because these IDEs focus on professional programmers and not on novice learners. This also affects the classrooms of the university “Pontificia Universidad Católica de Valparaíso (PUCV)” (Chile). The use of professional IDEs negatively affects the learning process of first-year students who face the development of the algorithms for the first time. One of the IDE widely used for teaching programming courses is Code::Blocks, which is a tool for professional developers. Through a heuristic and usability evaluation, we found that Code::Blocks has a complex user interface and a functional overload. Using these two findings, as well as recommendations given during these tests, we highlight the important aspects that an IDE for novice learners should have. Taking into account previous observations and state-of-the-art/practice of IDEs, a functional IDE prototype, named Incre-IDLE, is developed. In addition to Code::Blocks evaluations, this paper reports the results of a heuristic and usability evaluation applied to first-year students at PUCV about functionalities provided by Incre-IDLE. These results suggest that Incre-IDLE has a simple interface, is easy to install and use, and does not have functional overload (*i.e.*, spend a considerable amount of time learning the tool). Concretely, the results show that 66.7% of the students could complete tasks easily and 100% of them found the GUI intuitive. In terms of GUI, 83.3% considered the application interface “very simple”; and the text, concepts, and icons “very understandable” by 66.7%. The students also found the tool “motivating” (66.7%) or “very motivating” (33.3%). These results closely match the findings obtained by the heuristic evaluation of Incre-IDLE from the experts: 83.3% of them rated it as “useful” or “very useful”, and only a 16.7% rated it as

“useless”.

Keywords: IDE, First-year computer engineering students, Code::Blocks, Incre-IDLE

Categories: D.2.2, D.2.3, K.3

DOI: 10.3897/jucs.81329

1 Introduction

According to ACM and IEEE society, the computer curricula of any undergraduate degree in Computer Science include the programming courses [ACM/IEEE-CS, 2013]. Based on the fail rate, these courses are considered complicated in pedagogical terms, and it is generally the first time that students face the areas of algorithms and programming [Yadin, 2011, Ko et al., 2020, Figueroa et al., 2019]. These courses involve many different tasks and skills, such as learning how to structure algorithms, using an Integrated Development Environment (IDE), coding algorithms following a specific programming language syntax, using logical and abstract thinking, etc. The use of a professional IDE affects the process of teaching-learning in programming courses; there is evidence that the high failure rate of these courses is caused because there is no coherence between the tools used and the active learning strategies [Lahtinen et al., 2005].

IDEs, like Code::Blocks [Code::Block Team, 2022], are used by professors of the “Pontificia Universidad Católica de Valparaíso (PUCV)” (Chile) to teach programming courses in the first year of study of any career related to the Computer Science area. These IDEs are not focused on specialized programmers who are mostly developing an algorithm for the first time [Algaraibeh et al., 2020]. Based on teaching experience, it is considered that the use of professional environments harms learning in first-year students since numerous features are never used, and it distracts the students’ attention from learning [Kölling, 1999c]; however, these environments can simplify many processes like compiling, linking, and executing a program. Using the literature review and existing programming IDEs together, an heuristic and usability evaluations of Code::Blocks, we propose a first working IDE prototype, named Incre-IDLE. This proposal points out to remedy some of the problems that these students usually face. Incre-IDLE takes into account most of the aspects found and also provides students a good orientation regarding the use of buttons, texts, and comments, to solve tasks quickly and highlight errors before compilation; making it possible to recommend Incre-IDLE for use by novice learners and is useful for teaching courses. The productivity of Incre-IDLE shows how the application works and also the features that help users to learn programming and the features that should be added in other versions. In addition, this paper reports a case study on the use of Incre-IDLE through heuristic and productivity goal evaluations. For example, this report shows that 66.7% of the students could complete tasks, 100% found the GUI intuitive, and also found that Incre-IDLE is “motivating” (66.7%) or “very motivating” (33.3%).

In a few words, this paper’s methodology is comparing two IDEs for first-year students using heuristic and usability evaluations. We took a commonly used IDE, Code::Blocks, for first-year students from one Chilean university, and we carried out a heuristic and usability evaluation. Taking the evaluation results and review of existing IDEs, we then developed an IDE, Incre-IDLE, which should address the issues found in Code::Blocks. Finally, we carried out the same evaluations to compare both IDEs. This comparison allowed us to show that Incre-IDLE improved against Code::Blocks.

Paper roadmap. As this paper places a strong emphasis on evaluations of learning environments for first-year computer engineering students, we define a methodology that structures each section of this paper (Section 2). Following the proposed methodology, Section 3 identifies issues and recommendations related to the learning process of programming. Taking into account identified issues, recommendations, and Code::Blocks evaluations, Section 5 presents Incre-IDLE, a prototype IDE for these students. Section 6 presents different evaluations (*e.g.*, heuristic) of Incre-IDLE. Section 7 concludes this paper.

Availability. The prototype Incre-IDLE is available on <https://github.com/IncreIDLE/increidle>, and the questionnaire design for the Incre-IDLE evaluation is in the Google Sheets format on <https://github.com/pleger/increidle-results>, where the questions (with their evaluations) are in Spanish with an English translation. Finally, in the previous GitHub repository, a reader can find and try an executable installer for the operating system Microsoft Windows 8 and 10.

2 Methodology

To plan, design, and evaluate the proposal of Incre-IDLE, we use one methodology from a set of them available in [Kline and Seffah, 2005], particularly which evaluates an IDE for C++ using heuristic methods [Kline et al., 2002]. Figure 1 presents the methodology used in this paper to propose and evaluate Incre-IDLE, where we can see the sections that tackle the different components of this methodology. The methodology is mainly based on four stages: review, evaluate existing IDE (Code::Blocks), propose an IDE (Incre-IDLE), and evaluate and compare the proposal using heuristic and usability evaluations. As figure 1, each section of this paper, with their subsections, tackles a stage of this methodology. For the first step, we reviewed other similar IDEs available applied to the first-year students. In parallel to the first step, we evaluate the heuristic and usability of Code::Blocks. With the previous two outputs (Code::Blocks evaluations and existing IDE analysis), we develop a working prototype, named Incre-IDLE. Finally, we applied the same evaluation instruments to Incre-IDLE, where we added an evaluation of the application user-interface to measure if the user-interface might affect the usability evaluation by students.

2.1 Profile of Testers

As the dotted square in Figure 1 shows, the heuristic and usability evaluation for both IDEs use two different tester profiles: *advanced* for heuristic and *novice* for usability. As Table 1 shows, the advanced profile represents master and Ph. D. students, and the novice profile represents first-year students in a programming course. In the advanced profile, the testers already know a wide range of IDEs to develop that allows them to compare and give guidelines on how an IDE can help novice learners. Conversely, the

Evaluation	Profile	Type of Student	Number
Heuristic	Advanced	Master or Ph. D	6
Usability/Interface	Novice	Undergraduate	10

Table 1: Profiles of testers for evaluations

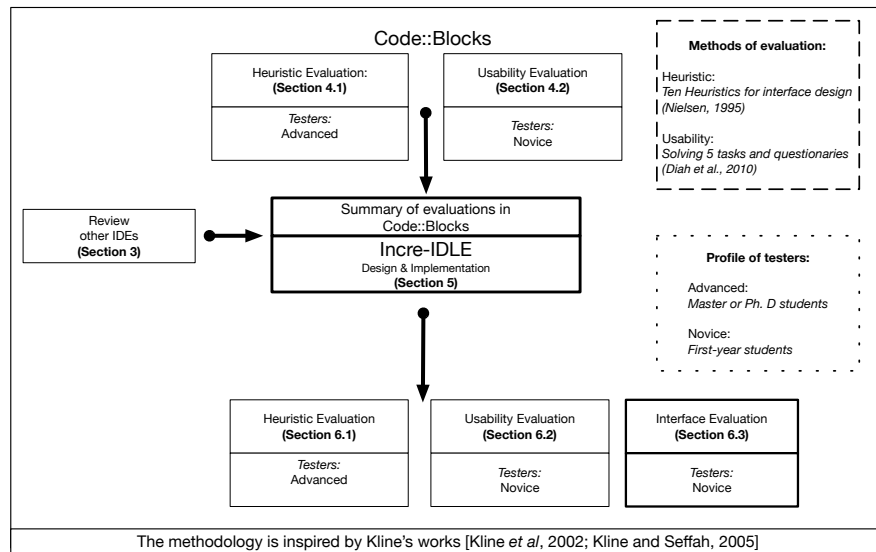


Figure 1: The methodology used in this study

testers with a novice profile give their feedback as real users of the IDEs: Code::Blocks and Incre-IDE. Regarding the selection of testers, we made a *call for participation* for each profile at the PUCV university. For the advanced profile, we made the call in the postgraduate programs, where six testers participated. Instead, the call to the novice profile was made in the first programming courses related to Computer Science programs, where ten testers accepted to participate.

2.2 Evaluations

Regarding the guidelines followed in carrying out the heuristic and usability evaluations in both IDEs, Code::Blocks and Incre-IDE, we used two different evaluation strategies (the dashed square in Figure 1). In the heuristic evaluation, we evaluate thirteen aspects that are related to the use of an IDE (Table 2). The definition of these aspects follow the description given in [Nielsen, 1995], which are evaluated by the testers with an advanced profile. For the usability evaluation, we followed the method used in [Diah et al., 2010], in which the testers first solve a set of tasks and then reply to a questionnaire related to these tasks. In our study, the testers with a novice profile first resolve five tasks (Table 3) using the IDE and then reply to a questionnaire (Table 4). For all questions, we used a Likert scale [Albaum, 1997] of five levels (*best* to *worst*), which level names were adapted to each question to help the student evaluations. Finally, the interface evaluation is one question (number 6 in Table 4) that groups a set of elements related to the interface.

3 First-year Students at Universities: Learning Process and Existing IDEs

This section highlights some studies that explain the aspects that inhibit learning in programming. It also discusses IDEs for educational purposes of understanding the

Id	Aspect	Description
H ₁	Engagement	The system must engage and motivate students
H ₂	Non-Threatening	The system does not seem threatening in appearance or behavior
H ₃	Minimal language redundancy	The programming language should minimize redundancy in their language constructs and libraries
H ₄	Learner-appropriate abstractions	The system must use abstractions that are at the appropriate level for the student and the task
H ₅	Consistency	The model, language, and presentation interface should be consistent internally and with each other
H ₆	Visibility	The user should always be aware of the status and progress of the system
H ₇	Secondary notations	The system must automatically provide secondary notations when useful
H ₈	Clarity	The presentation must maintain simplicity and clarity, avoiding visual distractions
H ₉	Human-centering syntax	Notation program must use syntax-centered human being
H ₁₀	Edit-order freedom	The interface should allow freedom of the user in the order you choose to work
H ₁₁	Minimal viscosity	The system should minimize viscosity in the entry and manipulation program
H ₁₂	Error-avoidance	Preference should be given to prevent errors on report them
H ₁₃	Feedback	The system must provide timely and constructive feedback

Table 2: Heuristic aspects used in the evaluation of IDEs

Id	Task	Description
1	Download and install	Tester should go to the Web page, download, and install the IDE
2	Open the IDE and check the initial interface	Tester should open the IDE and interacts with the initial interface
3	Create a new source file (c extension)	Tester should find the way to open a new file, select the location of the file, and give a name
4	Open an existing file, compile and execute	Tester should open a file that is already created, and execute it
5	Open a file with errors, compile and execute	Tester should be capable of finding the error in the file, fixing it, and executing the file without errors

Table 3: Tasks for testers

methodologies, common problems, and good practices in the learning-teaching process of programming courses. Finally, this section lists the difficulties that students have in learning and offers recommendations that an IDE should improve these difficulties.

3.1 Novice Learners in Programming

Teaching programming courses to novice students in the first-year represents a challenge for teachers and students who should deal with difficulties in the context of learning to program [Qian and Lehman, 2017]. This section describes the report of some experiences and lessons learned.

Difficulties of novice learners. A significant number of studies have been carried out to try to explain the difficulties of novice programmers; we briefly describe some of these studies. In [Lahtinen et al., 2005], the authors study learning difficulties in programming to support the development of material for basic programming courses (*CS1/CS2*). The study was applied to over 559 students and 34 professors from Universities in Germany, Iceland, Finland, Romania, and Latvia. The authors found difficulties in preparing

Id	Question	Possibles answers
1	Were you able to complete the task?	very easy, easy, borderline, hardly, very hardly
2	How oriented did you feel with the tool during the experiment?	very oriented, oriented, borderline, disoriented, very disoriented
3	How do you rate the application interface?	very simple, simple, borderline, complicated, very complicated
4	How motivated do you feel about the tool to learn to program?	very motivating, motivating, borderline, demotivating, very demotivating
5	Are the text, concepts, and icons of the environment interface understandable?	very understandable, understandable, borderline, complex, very complex
6	What do you think that news/messages are displayed within the Incre-IDLE application? <i>(Only for Incre-IDLE - Interface evaluation)</i>	very useful, useful, borderline, useless, very useless

Table 4: Questionnaire for testers

material and instructions that are adequate for each type of student in a class since programming courses are commonly quite extensive in concepts that we need at the same time (*e.g.*, i/o operations, variables, branches, loops). Therefore, previous difficulties cause high rates of students abandoning the classes. Regarding the contents of the courses, authors in [Lahtinen et al., 2005] highlight that some concepts are more difficult to learn because they require an understanding of larger entities in a program, which is also evidenced in similar articles like in [Soloway and Spohrer, 2013]. Students have problems understanding the basic concepts of programming, designing algorithms, dividing functionalities into methods, and finding errors in their programs. As with [Milne and Rowe, 2002], students also have difficulties with complicated concepts like pointers and memory management. Another similar study [Piteira and Costa, 2013] shows that authors use a group of 143 students of the University of Tabuk in basic programming courses to verify if they have similar problems to the students of different universities in the world. The study shows that students have the same difficulties as already mentioned. These common difficulties are: finding errors in programs, understanding abstract concepts (*e.g.*, recursion, pointers, and abstract data types), writing algorithms, methods usage, and syntax. Furthermore, the students complicated their experiences with the interpretations of the English language (a complex language for Japanese students) in a series of concepts used in the proposed problem statements. In the same line of previous studies, the authors in [McCracken et al., 2001] present a multi-national/institutional study of programming skill assessments of 216 first-year students from four universities. According to the study, the skills of students are mainly low, and students need previous experience (*e.g.*, in high school) to improve these skills.

Teaching introductory programming. In the learning process, we can find different proposals to start teaching programming. In [Koulouri et al., 2015], the authors claim that the programming proficiency of novice learners depends on the teaching approach. To validate, the authors propose three different approaches. The first one replaces Java with Python, the second one gives students only formative feedback, and the last one teaches the students how to solve problems before they take a programming course. Finally, the authors conclude that the early exposure and introduction to solving a problem is favorable regardless of the programming language used in the course. Unlike the previous paper, the authors in [Yadin, 2011] discuss three different *but mixed* approaches to teaching programming: the use of Python, visualization of program, and the use of individual assignments. The use of these approaches together helps reduce the number

of failing students by over 77%. Finally, the authors in [Utting et al., 2013] present the results of applying different strategies to teach programming. The results show that previous experience (*e.g.*, in high school) and/or recurrent feedback from teachers help the learning process.

3.2 Integrated Development Environments

To improve the learning process in novice learners, IDEs should provide a set of *specialized* features or *customize* existing ones [Algaraibeh et al., 2020, Kölling, 1999c]. For example, feedback or error messages that students receive and *read* when a compilation error appears in an IDE can significantly improve the learning process [Karvelas, 2019, Becker et al., 2018]. These IDEs are sometimes named Integrated Development and Learning Environment (IDLE). For this reason, it is possible to find studies such as in [Papadakis and Orfanakis, 2018], which compare the learning process in different programming environments that allow students to use visual interfaces to develop mobile applications. Given that IDEs are an important channel through which students experience programming [Dumas and Parsons, 1995], this section researches the IDEs that currently exist that focus on teaching programming in courses with novice students.

Visual Studio Code. The Microsoft company develops source code and multi-platform IDE: Visual Studio Code [Microsoft, 2022], which has around 2.6 Million monthly active users. According to the Slack survey 2021¹, this IDE is the most used by developers. Although this IDE is not specialized for novice learners, it is so versatile in terms of extensions and plugins that many users (and professors) use it daily.

Code::Blocks. This open-source IDE is for the programming language C/C++ and Fortran that works with multiple compilers [Code::Block Team, 2022]. Although the IDE is not adapted for novice learners, it is designed to be very lightweight, extensible, and fully configurable, implying that many institutions like PUCV (and users) use the IDE to teach. It is possible to find a number of studies carried out in Code::Blocks [Drumea, 2012, Delman et al., 2009, Soto and Figueroa, 2018]. In addition, Code::Blocks variants such as CodeLite² are available on the internet.

BlueJ. The design of BlueJ IDE [Kölling and Rosenberg, 1996, Kölling, 1999b] for Java, which is based on three fundamental principles that are simplicity, visualization, and interaction [Kölling, 1999a]. In addition, BlueJ supports regression testing by integrating with JUnit, 14 languages (*e.g.*, English). In addition, the IDE uses a *Code Pad* tool to instantly evaluate arbitrary expressions and phrases written in Java. In the body of literature, we can find studies about BlueJ, for example, in [Van Haaster and Hagan, 2004], the paper reports that the students preferred to use BlueJ instead of SDK of Java. Another study highlights that BlueJ has been used to carry out heuristic evaluations [Kölling and McKay, 2016].

Greenfoot. The IDE for Java, whose environment uses “*Interactive Visual World*”, where actors live and interact in created worlds, to develop games, simulators, and other graphic programs [Kölling, 2010]. The IDE’s user interface includes project management, auto-completion, syntax highlighting, and tools commonly used in most IDEs. The Greenfoot user interface has been designed to be used by beginners, so it focuses on a simple and ease to use this living world. Additionally, students can upload their projects and

¹ <https://insights.stackoverflow.com/survey/2021#integrated-development-environment>

² <https://codelite.org>

games to the platform's official website [Kölling, 2010], which can be used by any user who accesses it. Additionally, one can use Greenfoot as a first programming system for adolescents or older students. Greenfoot has also been used in the tests carried out for the preparation of the heuristic evaluations [Kölling and McKay, 2016].

DrJava. Among its main features are the intuitive user interface and the ability to interactively interpret Java code. The IDE allows programmers to develop, test, and debug programs interactively and incrementally. DrJava emphasizes the development of interactive software, offering a simple interface for writing code and the possibility of developing programs of a higher level of complexity [Rice University, 2022].

Python IDLE. This IDE is an integrated programming and teaching environment that is especially recommended for Python novice developers. This is because of the low complexity and the facilities that the IDE provides for teaching. Among Python IDLE features, we distinguish: syntax highlighting, multi-window text editing, auto-complete functions, integrated debugger with the possibility of step by step, breakpoints, and a of the call stack.

Zinjal. For C/C++, we find Zinjal, which was initially developed for educational use. This IDE currently includes different features for the development of more complex programs, without neglecting the simplicity of its interface for novice developers. With these features, a developer can use editing facilities, coding aids, an integrated debugging system, rapid development, auto-complete, template management, improved compilation results, integrated debugging, project management and a user interface in different languages like Spanish [Novara, 2010].

DrRacket. The graphical programming environment offers programmers facilities to develop in the Racket language. This language allows programmers to create new languages or dialects and to be used in a variety of environments, such as for teaching in computer engineering. In fact, this IDE has been developed in Racket. The simplicity of its interface stands out from this IDE, which is divided into three parts: toolbars (at the top), a definition panel where a user defines the program and an interactive panel to evaluate expressions (in the middle), and a programmer can see the status of IDE, such as the memory usage (at the bottom). DrRacket has been used in over 30 institutions to how to program³.

Table 5 summarizes and compares the previous IDEs. We can highlight two aspects in these features: adapted and interactive interface for novice learners.

3.3 Summary of Identified Difficulties and Recommendations

Difficulties and recommendations are useful as a frame reference to elaborate a proposal of an educational IDE for first-year university students. Based on the previous subsections (literature and existing IDEs), we next list these difficulties and recommendations associated with the learning-teaching programming process.

Difficulties:

1. Prepare material and instructions that are adapted to each type of student.

³ <https://github.com/racket/racket/wiki/Courses-using-Racket>

IDE	Language	Distinguish Features	Impact on Students
Visual Studio Code	Multiple languages	A very extensible and fully configurable.	Widely used for users for different multiple languages (over 2.6 Million monthly active users)
Code::Blocks	C/C++ and Fortran	A very extensible and fully configurable.	Widely used for students (over 46,886,918 downloads)
BlueJ	Java	Visual interaction of language components (e.g., classes)	They preferred this IDE instead of Java SDK
Greenfoot	Java	Live and visual interaction of objects	Adolescents and older students can use to develop videogames
DrJava	Java	Intuitive and interactive interface	Widely used for first-year students (over 4,332,375 downloads)
Python IDLE	Python	Adapted GUI for novice learners	It may be widely used because it is included by the Python distribution (over 20,000,000 downloads)
Zinjai	C/C++	Adapted GUI for novice learners	Widely used for first-year students that can speak Spanish (over 736,304 downloads)
DrRacket	Racket	Intuitive and interactive interface	Widely used for novice students in functional languages (over 30 institutes for teaching)

Table 5: Summary of different IDEs

2. Design programs that solve specific tasks and dividing functionalities into methods.
3. Find errors and problems in pieces of code.
4. Understand abstract concepts like pointers and memory management.
5. Apply mathematics and English knowledge.
6. Work in a group. This aspect may differ from the country culture [Lahtinen et al., 2005].
7. Apply theoretical concepts in practice.
8. Formulate clear and precise questions about programming.
9. Understand (quickly) compilation errors, particularly syntax errors.

Recommendations:

1. Apply the basic programming concepts together.
2. Explain contents in a practical way and with examples.
3. Teach materials that focus on solving a problem, rather than just representing concepts.
4. Use Parsons-Puzzles [Parsons and Haden, 2006] inspired questions to develop abstraction skills.

4 Heuristic and Usability Evaluations of Code::Blocks

This section presents the results obtained from the heuristics and usability tests of Code::Blocks.

4.1 Heuristic Evaluation

Using postgraduate students (Table 1), a heuristic evaluation was carried out. The methodology consisted of observing and inspecting the system based on its interface and functionalities. Following the aspects described in [Nielsen, 1995] and shown in Table 2,

we evaluated thirteen heuristics (H_i): Engagement (H_1), Non-Threatening (H_2), Minimal language redundancy (H_3), Learner-appropriate abstractions (H_4), Consistency (H_5), Visibility (H_6), Secondary notations (H_7), Clarity (H_8), Human-centering syntax (H_9), Edit-order freedom (H_{10}), Minimal viscosity (H_{11}), Error-avoidance (H_{12}), and Feedback (H_{13}).

Each tester identified the number of problems per heuristic, where H_2 and H_8 showed the highest number of problems (Figure 2). These thirteen heuristics are related to the interface of the system, indicating that a new potential IDE, like our proposal, diminishes or completely eliminates these deficiencies.

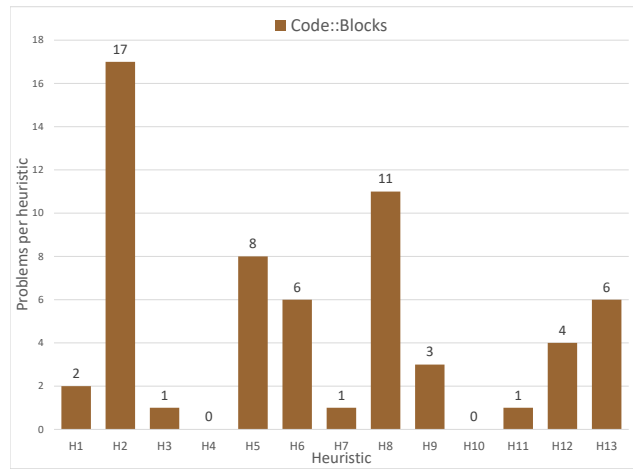
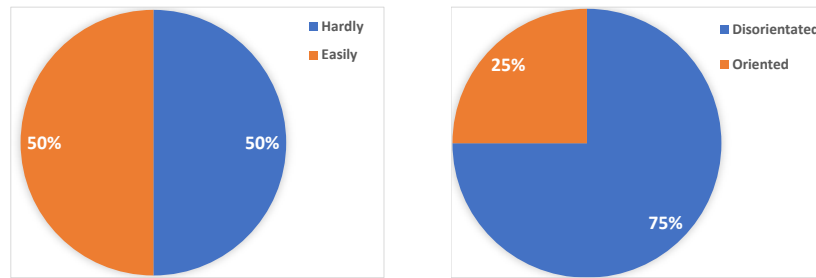


Figure 2: Number of problems found per heuristic

Although it is impossible to identify *all* potential issues in Code::Blocks, we can find the most critical problems that could affect students, thanks to the heuristic evaluation. After the evaluation, the testers were independently requested to classify the problems in terms of their severity and frequency: “Catastrophic”, “Major”, “Minor”, “Cosmetic”, and “Not a Problem”. The classification helps find some relevant aspects that came to light in the analysis. One of those is the lack of a tutorial to use the different functionalities offered by the tool, meaning that the best benefit of the IDE cannot be obtained. Furthermore, without a user guide, it is confusing for users to have so many buttons and tools at their interface, and that they do not present any information. This IDE has several positive aspects that can help in learning programming through this environment. These aspects are icons with *tooltip*, indicators with colors the lines of code that have been modified since the last compilation of the program, contextual help display when starting to write certain functions, and use colors in the text fonts to highlight reserved words. Moreover, the Code::Blocks IDE also checks symbol balance in the source code (symbols not closed), auto-completes when parenthesis are open, hints to auto-completion when typing in the source code, and formatting automatically when writing a set of instructions



(a) Were you able to complete the task set? (b) How oriented did you feel with the tool during the experiment?

Figure 3: Questions 1 and 2 (Code::Blocks)

inside a pair of braces. However, many beneficial functionalities and the intuitive layout of the interface are overshadowed by the complexity of the system. It is necessary to simplify the functionalities that are available in Code::Blocks for novice learners since the number of pre-installed plugins complicates the task of learning programming with Code::Blocks.

This heuristic evaluation, which indicates errors and benefits of Code::Blocks, becomes an input to define our proposal: Incre-IDLE; which is going to be evaluated using this kind of evaluation.

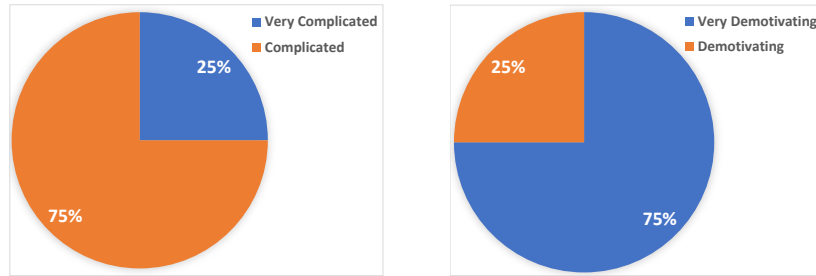
4.2 Usability Evaluation

For the usability evaluation, we followed the method used in [Diah et al., 2010], where the testers first solve a set of tasks and then reply to a questionnaire related to these tasks. The testers with a novice profile (Table 1) have to develop five simple tasks (Table 3). After completing the tasks, the students were asked to fill out a form with questions (Table 4). We evaluated Code::Blocks based on these five questions. Figure 3 shows how easy it is to complete the task and how oriented a student was during the use of the IDE. Figures 4 and 5 show how the students considered the interface of Code::Blocks, the icons, text, and also if they would use the IDE as a tool for learning programming. The results show the close relationship between the behavior of the students when executing the tasks and the heuristic evaluations, which is a motivation to develop an IDE that improves these evaluations.

5 Incre-IDLE: An IDE for First-Year Students

Using difficulties, recommendations, and Code::Blocks evaluations, we have proposed Incre-IDLE as a functional prototype of an IDE for first-year students.

Incre-IDLE focuses on students of Computer Civil Engineering at PUCV. First, Incre-IDLE includes some *addons* such as an administration panel and a website. The administration panel allows users to report *logs* and *bugs*, which are used to send review communications, news, and administer registered students (who use the application).



(a) How do you rate the application interface?

(b) How motivated do you feel about the tool to learn programming?

Figure 4: Questions 3 and 4 (Code::Blocks)

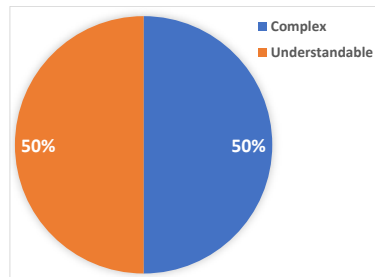


Figure 5: Are the texts, concepts, and icons of the environment interface understandable?
Question 5 (Code::Blocks)

The website allows users to download Incre-IDLE itself in addition to additional plugins, review course announcements, and review tutorials. Figure 6 shows a graphic representation that specifies components and actors that interact in our proposal. Second, the user interface is a fundamental theme for the correct project development. Figure 7 shows components of welcome and editor screens. Note that application messages are in Spanish because it is the official language in Chile.

5.1 Implementing Incre-IDLE

Various IDEs offer the possibility of extending their functionalities to be able to implement a new personalized tool, such as Eclipse, Atom, Brackets, or Visual Studio Code⁴. These IDEs were reviewed to develop Incre-IDLE. As a result, we chose the Atom because it is a multi-platform open and available source code editor, available for macOS, Linux, and Windows. Apart from the C language support, the installation of *packages* allows developers to expressively customize the text editor, since the vast

⁴ <http://www.eclipse.org>, <http://atom.io>, <http://brackets.io>, <http://code.visualstudio.com>. Last visited: 12/04/2022.

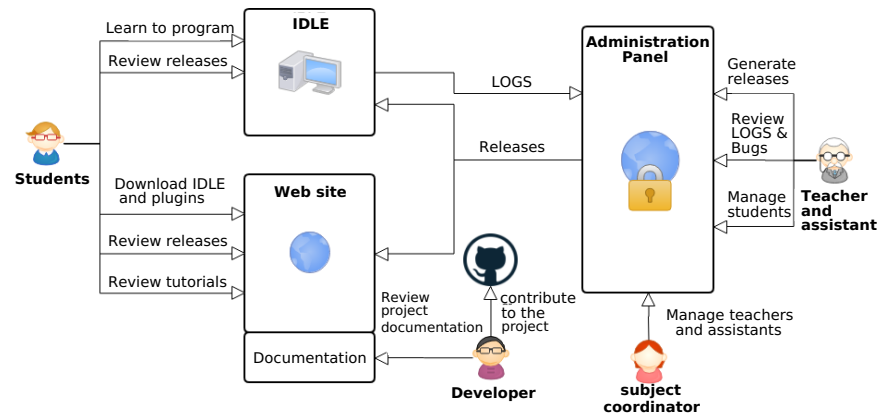


Figure 6: Outline solution, which shows components and actors that interact in Incre-IDLE

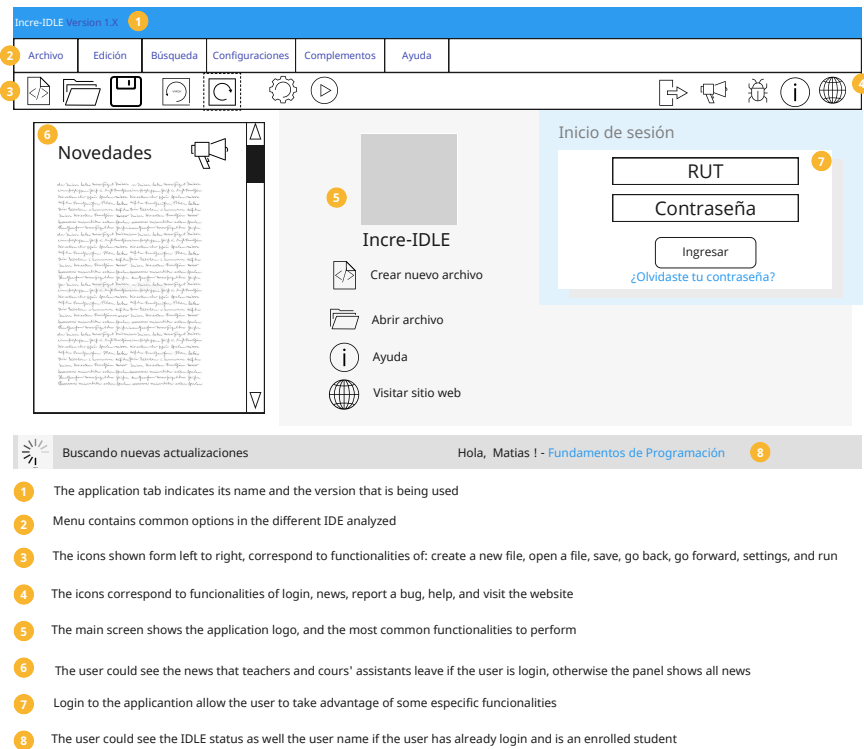


Figure 7: Incre-IDLE start prototype

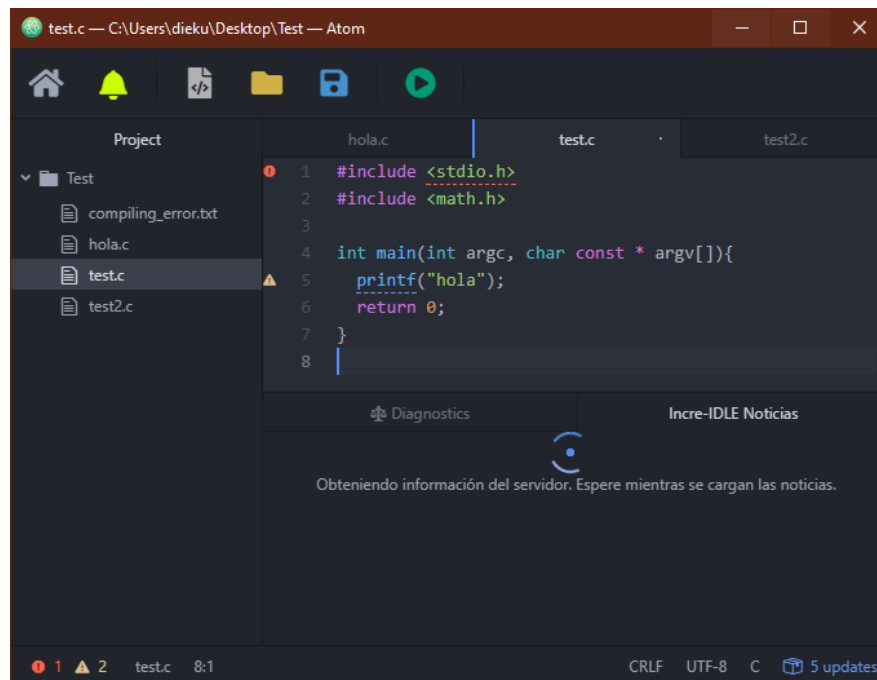


Figure 8: Incre-IDE news

majority of the Atom core is implemented through these packages. Additional packages are required to install in order to provide a graphical interface, functionality, and support for the C language. Finally, we chose Atom because it is frequently being updated by the community⁵. All modified packages are available on the Incre-IDE repository, which is available for future improvements.

When installing, the default packages were left because this same editor may be used by students in future courses. The benefits of programming in C language with Atom are, for example, highlighting in real-time the lines of code that have syntax errors, unlike Code::Blocks that requires compiling the code previously. Incre-IDE will *Compile* and *Execute* the written code and display a toolbar at the top of the editor.

Add-ons such as news from the teachers, assistants, and coordinators of the course (see Figure 8) are administered through an administration panel. This feature helps students send logs, bugs, and stay informed of the latest news from the course. There are three user profiles that can access this administration panel. The first is the Course coordinator, who administers configurations and accounts that can access this administration panel. The second and third are the Teacher and Teaching Assistant respectively, who administer news of the subjects that the coordinator has assigned them (Figure 9).

The Incre-IDE website was developed using Jekyll⁶, which is a simple static website generator providing blog features. This simplifies the publication of updates regarding IDE. This website provides students with the most relevant information about the tool,

⁵ <https://atom.io/releases>. Last visited: 14/04/2022.

⁶ <https://jekyllrb.com>. Last visited: 13/04/2022.

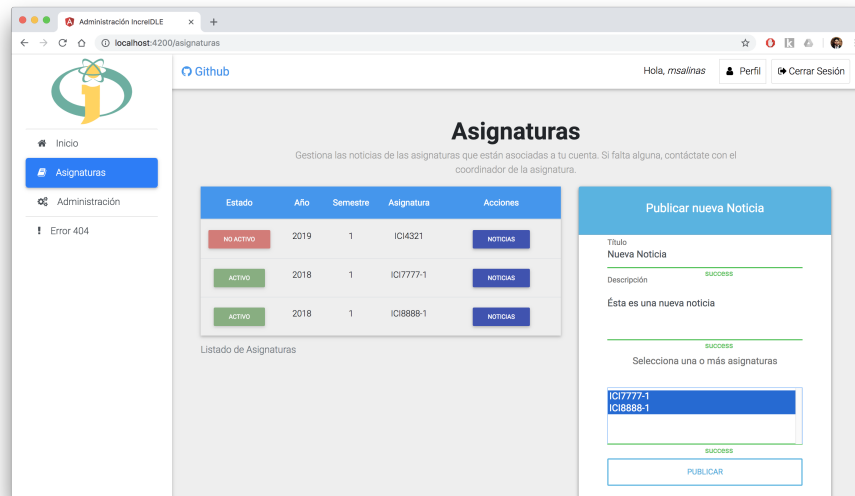


Figure 9: News management of the administration panel

such as: the download, tutorials, links of interest, among others. In addition, the website shows sections for news on each of the active subjects and tutorials on how to develop new plugins for this IDE.

6 Evaluations of Incre-IDLE

We have proposed Incre-IDLE, an IDE that aims to address the issues found in the heuristic and usability evaluations of Code::Blocks. To know if Incre-IDLE can help improve the learning-teaching process in first-year students, we carried out the same kind of evaluations used in Code::Blocks. As Table 1 shows, we selected different students for each evaluation, which followed the same participant configuration used for the Code::Blocks evaluations (Section 4). In addition, we added the interface evaluation to allow us to measure its graphic/visual design.

6.1 Heuristic Evaluation

We evaluated Incre-IDLE with the same heuristics that previously evaluated Code::Blocks. We analyzed the most critical problems and focused on those to develop Incre-IDLE. We asked the same testers to classify the issues found in Incre-IDLE.

In Figure 10, we compared the results and noticed that Incre-IDLE simplifies the functionalities and avoids distractions (H_8). Incre-IDLE helped with windows and tips that are useful for novice learners. Regarding the Consistency heuristic (H_5), Incre-IDLE is very clear in the installer process and has a tutorial on the website to guide the user in this process. Compared to Code::Blocks, Incre-IDLE encourages students to explore the IDE and proves how the code works without worrying about breaking the system (Non-Threatening (H_2)). Although Incre-IDLE presents some problems that should be solved

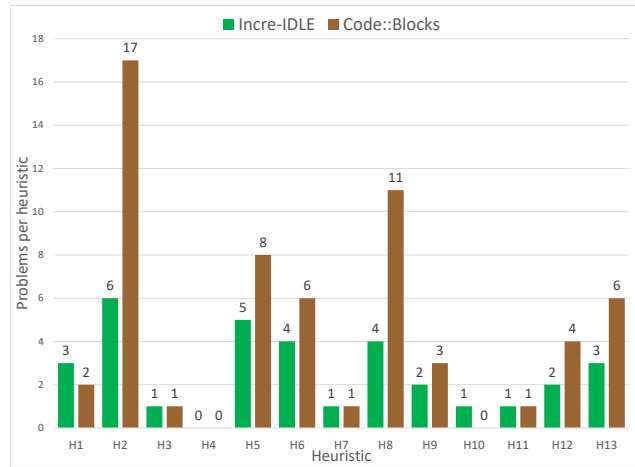


Figure 10: Problems per heuristics found in Incre-IDLE and Code::Blocks

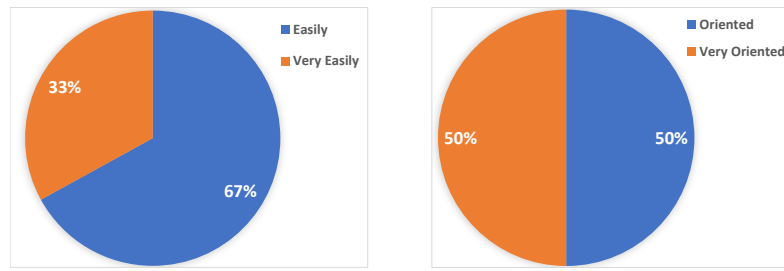
(e.g., Engagement (H_1)), this first version decreases the problems found in Code::Blocks and helps students to learn to programming.

6.2 Usability Evaluation

Following the usability evaluation used in [Diah et al., 2010], we evaluated Incre-IDLE to know whether its design helps in the learning process or not. Testers performed the same tasks used with Code::Blocks (Table 3). The results with the implementation in Incre-IDLE were very encouraging. This is because the participants achieved 100% of effectiveness in obtaining the proposed tasks as Table 6 shows. The average time improved compared to the participants of Code::Blocks. This reduction is directly related to the simplicity perception of the participants using Incre-IDLE, since the fulfillment of the proposed tasks were carried out without major inconveniences thanks to the correct distribution of the elements in the designed interface.

	Average Code::Blocks	Average Incre-IDLE	Difference	Reduction Percentage
Task 1	1 min. 50 secs.	25 secs.	1 min. 25 secs.	-77.3%
Task 2	18 secs.	11 secs.	7 secs.	-38.9%
Task 3	1 min. 10 secs.	29 secs.	41 secs.	-58.6%
Task 4	1 min. 53 secs.	15 secs.	1 min. 38 secs.	-86.7%
Task 5	45 secs.	13 secs.	32 secs.	-71.1%

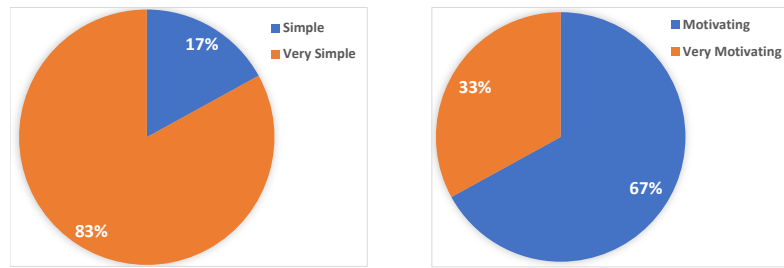
Table 6: Comparison of average time



(a) Were you able to complete the task set?

(b) How oriented did you feel with the tool during the experiment?

Figure 11: Questions 1 and 2 (Incre-IDLE)



(a) How do you rate the application interface?

(b) How motivating do you feel the tool to learn programming?

Figure 12: Questions 3 and 4 (Incre-IDLE)

Testers were also asked to answer a short questionnaire of five questions, expressing their opinions regarding the tasks that they had to fulfill, and what facilities the evaluated tool provided them to be able to fulfill them. Figure 11 shows the results of the first and second questions, which are oriented to the difficulty in using the tool. We can see that the tasks could be “(very) easily” carried out 100% (Figure 11a). The participants felt oriented in the use of Incre-IDLE since, at all times, they manifested the simple interface of Incre-IDLE, and the usefulness of the auxiliary elements that this tool presents (Figure 11b). Figure 12 shows the perception regarding the proper distribution of the elements in the interface, in addition to reducing the functionalities available to its users. Additionally, the participants responded that the tool for learning programming is “motivating” (Figure 12b). Finally, the last question of the questionnaire seeks to know if the students think that the texts, concepts, and icons of the application user interface are understandable. As regards this last question, Figure 13 shows that the participants were able to understand the elements that were presented in the application interface.

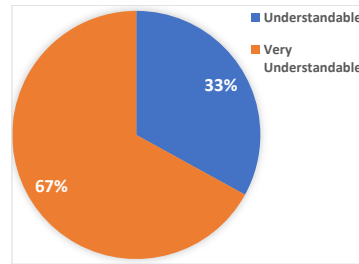


Figure 13: Are the texts, concepts, and icons of the environment interface understandable?
Question 5 (Incre-IDLE)

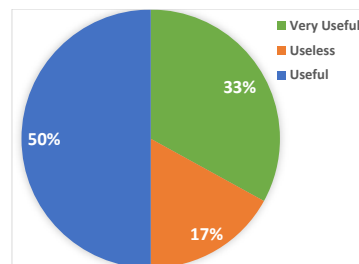


Figure 14: What do you think that news/messages are displayed within the Incre-IDLE application? (Only for Incre-IDLE - Interface evaluation)

6.3 Interface Evaluation

Incre-IDLE is a functional prototype, which includes features to program, extends the IDE itself, and associated tools to help in the learning process. We evaluated three aspects of the prototype: the installation process (starting its use), the Incre-IDLE application itself (its use), and the administration panel (its administration/maintenance). The testers, first-year students, were then shown the application interface, button layout, and functionalities. At the end, the testers were given a series of questions regarding their general perception of the system to know from their viewpoints, where positive and negative aspects they found. The survey of the results was presented along with a brief analysis regarding the installation process, the Incre-IDLE application, and final perception.

6.3.1 Installation and Application

The testers were shown the process that students must go through to install Incre-IDLE. Favorably, all testers feel that the installation was not complicated and that all the steps to be taken to complete the installation were understood. However, some testers gave some additional suggestions such as: “At the end of the installation, the Incre-IDLE website could be opened with the *first steps* tutorial” and “the website should also show videos with the installation process”.

Students/testers were asked their opinions about news and messages from teachers that could be viewed within the software application, where Figure 14 shows that the majority of respondents find this functionality “useful”. However, only one tester found it “useless”, arguing that messages were not fully understandable. Finally, the testers were asked if they would change any element, text, or functionality of the IDE:

- Some menus look “useless”, such as *Packages* and *Nuclide*.
- In the upper bar of the application, the options are accompanied by letters, for example, *File (F)*, and the testers do not clearly understand why these options are shown like this.
- The notification message is not very clear to a novice programmer who has no prior knowledge of the English language. This message should be translated into Spanish, and it would be useful to enable an option to review recently opened files.

The testers would recommend this tool to a student just beginning to study programming because:

- The user interface is user-friendly and shows useful information when compiling.
- The homepage of the Incre-IDLE website is a friendly interface because this website does not intimidate the user and is easy to install, allowing the focus to study C language directly and not end up getting frustrated trying to work with the IDE.
- An inexperienced and novice user will not suffer from a saturation of elements or an excess of complex and cumbersome options that tend to be frustrating.
- Incre-IDLE is user-friendly, in addition to many other tools that currently exist on the market. It is oriented towards freshmen, and our proposal does not show irrelevant content that could confuse and affect the learning experience.
- Functionalities and the distribution of options are simple and quick to understand.

6.4 Summary

The results of the heuristic evaluation show that testers consider the user interface *adequate* for novice learners according to the value scale defined in [Nielsen, 1995]. In addition, the testers consider Incre-IDLE adequate for the needs of novice learners compared to Code::Blocks, which is more for professional users (Figure 10). We exemplify the previous affirmation using Figure 15, which shows the percentage of the thirteen aspects in the heuristic evaluation where an IDE had fewer problems (using the heuristic score) compared to the other IDE. Apart from showing a draw of 30.8% (4 aspects) between both IDEs, the figure shows that Incre-IDLE have fewer problems (53.8%) than Code::Blocks (15.4%).

The results of the usability evaluation are *successful* because the obtained results affirm that the first-year students satisfactorily accepted the developed prototype (Figures 11, 12, and 13). Regarding the design, we found that Incre-IDLE has a balance between the functionalities to avoid visual distractions and does not complicate the use of the tool. Concretely, we think that Incre-IDLE, in general, has a simple user interface because 83.3% of the students consider the use of the tool “very simple” while 16.7%

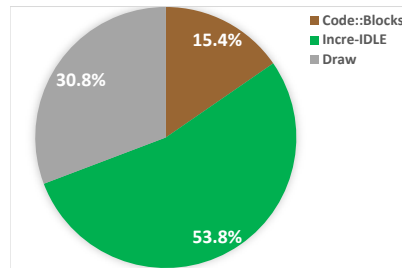


Figure 15: Percentage of the thirteen aspects in the heuristic evaluation where an IDE had fewer problems (using the score) compared to the other IDE

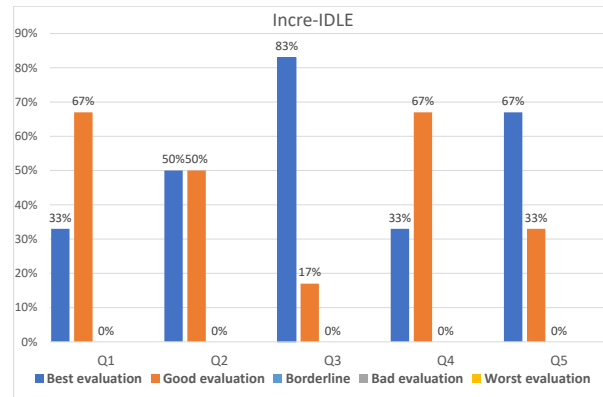
claimed it to be just “simple” (Figure 12). Incre-IDLE allows students to reduce performance time for a simple task from 38.9% to 86.7% compared to the time used with Code::Blocks (Table 6). To illustrate the previous affirmations, Figure 16 summarizes the usability evaluation between both IDEs using a Likert scale [Albaum, 1997] of five levels. It is easy to see that Incre-IDLE have better evaluations (near to “best”) than Code::Blocks.

Regarding the application interface evaluation, the testers found that the Incre-IDLE’s interface is “very useful” (33%) and “useful” (50%) as Figure 14 shows; however, these testers proposed a list of changes that could be carried out in Incre-IDLE. The changes are related to unnecessary extra information that Incre-IDLE shows. Finally, testers comment that a simple interface for the Incre-IDLE’s web page does not intimidate novice learners.

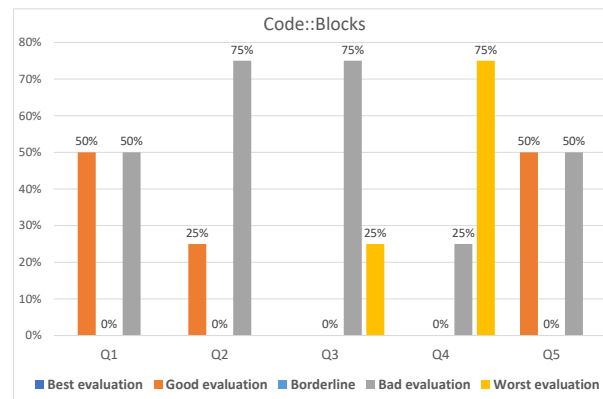
6.4.1 Problems to Solve

Although the results are favorable in terms of the usability for Incre-IDLE, there are some elements that must be solved and improved:

- There is an error trying to create a file if the user has not opened a folder/project. It is necessary to analyze which solution is adequate for this problem.
- When users try to create a file, the message that is delivered to users is not entirely understandable by novice learners.
- There is an error console when a program fails to be compiled, which can only be accessed manually. This console can be quite useful for Incre-IDLE users, so every time a program is compiled, the error console should be displayed.
- When users compile a program with errors, a notification is displayed in the upper right of the application. The students have stated that the active time of the notification is too short, and the message that they want to deliver cannot be read, so the active time of the notification should be increased.



(a) Incre-IDLE



(b) Code::Blocks

Figure 16: Summary of the usability evaluation, five questions, for both IDEs

7 Conclusions and Future Work

This article presents a prototype IDE for first-year students, named Incre-IDLE, which aims to address different issues in the teaching-learning process of programming courses. To propose our IDE, we carried out a heuristic evaluation to Code::Blocks (a widely used IDE). Incre-IDLE considers the difficulties found in Code::Blocks evaluations and the literature. The requirements and user interface presented in this paper correspond to the first functional version of Incre-IDLE.

The developed IDE completely focuses on novice learners, considering the students'

own needs. We expect that in the near future, this project will reach the classrooms of first-year students as soon as possible, so that they can enjoy a pleasant experience when using Incre-IDLE. The interface, heuristic, and usability evaluations of the Incre-IDLE allowed us to observe that the proposed idea can be used by any user without any programming knowledge. In addition, we think Incre-IDLE allows professors to organize the teaching courses of these courses between different teachers in an easy way.

Future Work

Throughout the development of the study, we have identified a set of improvements that a future study might carry out. We classified these improvements into two groups: evaluation design and feature implementations.

Evaluation Design. We followed one methodology of a set that is available in [Kline and Seffah, 2005, Kline et al., 2002] to carry out this study. This methodology includes heuristic and usability evaluations. For the heuristic evaluation, we used the ten usability heuristics for design interfaces [Nielsen, 1995]. For the usability evaluation, we asked students to resolve five (simple) tasks and reply to a questionnaire. In the current application of this methodology, we found the following limitations:

- The design of these tasks and questionnaires did not follow a methodology that explicitly relates each question with to activity of the tasks. In a future study, following the recommendations of the questionnaire design for research [Creswell, 2014], we should consider defining a set of *study variables* to allow researchers to design and relate the questionnaire and tasks.
- As the study mentions, the testers solve only five simple tasks. A larger number of tasks can bring benefits to this research. First, make a closer relationship between the questionnaire’s purpose and tasks. Second, specify different difficulty levels for tasks, where levels can be related to programming skills of the testers.

All testers that participated were men, implying that this study does not show whether Incre-IDLE is useful for women. Likewise other researches [Akinola, 2015], it is valuable to consider evaluations with significant participation of both genders, and compare them.

Finally, although the studies presented by Soloway and Spohrer might be considered old [Spohrer and Soloway, 1986, Soloway and Spohrer, 2013], these studies are still present; for example, in the book “Studying the Novice Programmer” whose first edition was in 1986, we can find an edition in 2013 [Soloway and Spohrer, 2013]. Hence, we might adapt and improve our methodology based on the previous authors’ studies.

Feature Implementations. There are still challenges for Incre-IDLE in terms of implementations. For example, the missing functionalities should be implemented, as well as improving the complements that are currently used in the first functional prototype, such as the customization and reduction of auto-completion options of code. Additionally, it is necessary to update the website with the new functionalities that are included in Incre-IDLE, as well as to develop, in the administration panel, a monitoring module (*logs* and *bugs*) of the student behavior when Incre-IDLE is in use.

References

- [ACM/IEEE-CS, 2013] ACM/IEEE-CS (2013). Computer Science Curricula 2013. Technical report, ACM Press and IEEE Computer Society Press.
- [Akinola, 2015] Akinola, S. (2015). Computer Programming Skill and Gender Difference: An Empirical Study. *American Journal of Scientific and Industrial Research*, 7(1):1–9.
- [Albaum, 1997] Albaum, G. (1997). The Likert Scale Revisited. *Market Research Society Journal*, 39(2):1–21.
- [Algaraibeh et al., 2020] Algaraibeh, S., Dousay, T., and Jeffery, C. (2020). Integrated Learning Development Environment for Learning and Teaching C/C++ Language to Novice Programmers. In *IEEE Frontiers in Education Conference (FIE)*, pages 1–5, Uppsala, Sweden.
- [Becker et al., 2018] Becker, B., Goslin, K., and Glanville, G. (2018). The Effects of Enhanced Compiler Error Messages on a Syntax Error Debugging Test. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 640–645, Maryland, USA.
- [Code::Block Team, 2022] Code::Block Team (2022). Code::Blocks: A Free IDE for C/C++. Version 20.03. <https://www.codeblocks.org>. Visited on 18/07/22.
- [Creswell, 2014] Creswell, J. (2014). *Research Design: Qualitative, Quantitative and Mixed Methods Approaches*. Sage.
- [Delman et al., 2009] Delman, A., Goetz, L., Langsam, Y., and Raphan, T. (2009). Development of a System for Teaching C/C++ Using Robots and Open Source Software in a CS1 Course. In *Frontiers in Education: Computer Science & Computer Engineering (FECS)*, pages 141–146, Las Vegas, USA.
- [Diah et al., 2010] Diah, N. M., Ismail, M., Ahmad, S., and Dahari, M. K. M. (2010). Usability Testing for Educational Computer Game Using Observation Method. In *International Conference on Information Retrieval & Knowledge Management (CAMP)*, pages 157–161, Mara, Malaysia.
- [Drumea, 2012] Drumea, A. (2012). Education in Development of Electronic Modules Using Free and Open Source Software Tools. *Hidraulica*, (3-4/2012).
- [Dumas and Parsons, 1995] Dumas, J. and Parsons, P. (1995). Discovering the Way Programmers Think About New Programming Environments. *Communications of the ACM*, 38(6):45–56.
- [Figueroa et al., 2019] Figueroa, I., Jiménez, C., Allende-Cid, H., and Leger, P. (2019). Developing Usability Heuristics with PROMETHEUS: A Case Study in Virtual Learning Environments. *Computer Standards & Interfaces*, 65:132–142.
- [Karvelas, 2019] Karvelas, I. (2019). Investigating Novice Programmers’s Interaction with Programming Environments. In *Proceedings of ACM Conference on Innovation and Technology in Computer Science Education*, pages 336–337, Aberdeen Scotland, United Kingdom.
- [Kline and Seffah, 2005] Kline, R. and Seffah, A. (2005). Evaluation of Integrated Software Development Environments: Challenges and Results from Three Empirical Studies. *International Journal of Human-Computer Studies*, 63(6):607–627.
- [Kline et al., 2002] Kline, R., Seffah, A., Javahery, H., Donayee, M., and Rilling, J. (2002). Quantifying Developer Experiences via Heuristic and Psychometric Evaluation. In *Proceedings IEEE Symposia on Human Centric Computing Languages and Environments*, pages 34–36, Arlington, USA.
- [Ko et al., 2020] Ko, A., Oleson, A., Ryan, N., Register, Y., Xie, B., Tari, M., Davidson, M., Druga, S., and Loksa, D. (2020). It is Time for More Critical CS Education. *Communications of the ACM*, 63(1):31–33.
- [Kölling, 1999a] Kölling, M. (1999a). Teaching Object Orientation with the Blue Environment. *Journal of Object-Oriented Programming*, 12(2):14–23.

- [Kölling, 1999b] Kölling, M. (1999b). *The Design of an Object-Oriented Environment and Language for Teaching*. PhD thesis, Basser Department of Computer Science, University of Sydney.
- [Kölling, 1999c] Kölling, M. (1999c). The Problem of Teaching Object-Oriented Programming, Part 2: Environments. *Journal of Object-Oriented Programming*, 11(9):6–12.
- [Kölling, 2010] Kölling, M. (2010). The Greenfoot Programming Environment. *ACM Transactions on Computing Education*, 10(4):1–21.
- [Kölling and McKay, 2016] Kölling, M. and McKay, F. (2016). Heuristic Evaluation for Novice Programming Systems. *ACM Transactions on Computing Education*, 16(3):1–30.
- [Kölling and Rosenberg, 1996] Kölling, M. and Rosenberg, J. (1996). An Object-Oriented Program Development Environment for the First Programming Course. In *Proceedings of SIGCSE Technical Symposium on Computer Science Education*, pages 83–87, Philadelphia, USA.
- [Koulouri et al., 2015] Koulouri, T., Lauria, S., and Macredie, R. (2015). Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. *ACM Transactions on Computing Education*, 14(4):1–28.
- [Lahtinen et al., 2005] Lahtinen, E., Ala-Mutka, K., and Järvinen, H.-M. (2005). A Study of the Difficulties of Novice Programmers. In *Proceedings of Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 14–18, Caparica, Portugal.
- [McCracken et al., 2001] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolkant, Y. B.-D., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001). A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, pages 125–180, Canterbury, United Kingdom.
- [Microsoft, 2022] Microsoft (2022). Visual Studio Code: A Free IDE for Multiple Languages. Version 1.63. <https://code.visualstudio.com>. Visited on 18/07/22.
- [Milne and Rowe, 2002] Milne, I. and Rowe, G. (2002). Difficulties in Learning and Teaching Programming-Views of Students and Tutors. *Education and Information Technologies*, 7(1):55–66.
- [Nielsen, 1995] Nielsen, J. (1995). Ten Usability Heuristics for User Interface Design. <https://www.nngroup.com/articles/ten-usability-heuristics>. Updated: Nov. 15, 2020 and visited on 12/04/2022.
- [Novara, 2010] Novara, P. (2010). Fundamentos de Programación –Anexo 1: Introducción a las Herramientas de Desarrollo. <http://zinjai.sourceforge.net/Anexo1.pdf>. Visited on 12/04/2022.
- [Papadakis and Orfanakis, 2018] Papadakis, S. and Orfanakis, V. (2018). Comparing Novice Programming Environments for use in Secondary Education: App Inventor for Android vs. Alice. *International Journal of Technology Enhanced Learning*, 10(1/2).
- [Parsons and Haden, 2006] Parsons, D. and Haden, P. (2006). Parson’s Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses. In *Proceedings of Australasian Conference on Computing Education*, pages 157–163, Virtual SA Australia.
- [Piteira and Costa, 2013] Piteira, M. and Costa, C. (2013). Learning Computer Programming: Study of Difficulties in Learning Programming. In *Proceedings of International Conference on Information Systems and Design of Communication*, pages 15–24, Marioka, Japan.
- [Qian and Lehman, 2017] Qian, Y. and Lehman, J. (2017). Students’ Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Transactions on Computing Education*, 18(1):1–24.
- [Rice University, 2022] Rice University (2022). DrJava: A Lightweight Pedagogic Environment for Java. Version 20190813. <http://www.drjava.org>. Visited on 18/07/22.

- [Soloway and Spohrer, 2013] Soloway, E. and Spohrer, J. (2013). *Studying the Novice Programmer*. Interacting with Computers Series. Psychology Press.
- [Soto and Figueroa, 2018] Soto, M. and Figueroa, I. (2018). Heuristic Evaluation of Code::Blocks as a Tool for First Year Programming Courses. In *37th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–8, Santiago, Chile.
- [Spohrer and Soloway, 1986] Spohrer, J. C. and Soloway, E. (1986). Novice Mistakes: Are the Folk Wisdoms Correct? *Communications of the ACM*, 29(7):624–632.
- [Utting et al., 2013] Utting, I., Sorva, J., Wilusz, T., Tew, A. E., McCracken, M., Thomas, L., Bouvier, D., Frye, R., Paterson, J., Caspersen, M., and Kolikant, Y. B.-D. (2013). A Fresh Look at Novice Programmers’ Performance and Their Teachers’ Expectations. In *Proceedings of the ITiCSE Working Group Reports Conference on Innovation and Technology in Computer Science Education-Working Group Reports*, pages 15–32, Canterbury England, United Kingdom. ACM Press.
- [Van Haaster and Hagan, 2004] Van Haaster, K. and Hagan, D. (2004). Teaching and Learning with BlueJ: An Evaluation of a Pedagogical Tool. *Issues in Informing Science and Information Technology*, 1:455–470.
- [Yadin, 2011] Yadin, A. (2011). Reducing the Dropout Rate in an Introductory Programming Course. *ACM Inroads*, 2(4):71–76.