International Journal of Advanced Multidisciplinary Application | IJAMA
Volume 2 Issue 1 Jan 2025
ISSN No: 3048-9350

Enhancing Programming Learning Through the Integration of Emerging Technologies and Established Tools

Amadou S. Bah[1], Fatimah J. Ceesay[2], and Ousman B. Jallow[3]
[1]Department of Computer Science, University of Gambia, Banjul, Gambia
[2]Department of Electrical Engineering, University of Gambia, Banjul, Gambia
[3]Department of Information Technology, University of Gambia, Banjul, Gambia

***Abstract-** In recent years, a variety of tools have been proposed to alleviate the challenges faced by students learning programming. Our group has contributed significantly to this field with the development of tools such as VIP, SICAS, OOP-Anim, SICAS-COL, and H-SICAS. While these tools have shown some positive results, their impact on reducing the difficulties of weaker students has not been as substantial as expected. These students often require more robust support to motivate and engage them in learning, both inside and outside the classroom. With the advent of new technologies, we believe that the integration of various solutions can play a crucial role in overcoming these challenges. This paper evaluates the features, strengths, and weaknesses of the tools developed by our group, identifying areas for improvement. Based on these insights, we propose the development of a new environment that integrates multiple pedagogical approaches, resources, tools, and technologies to support programming learning. The new environment, currently under development, will enable students to review content and lessons through video and screen captures. It will also incorporate collaborative task support, fostering teamwork and collaboration among students. Additionally, the platform will facilitate the creation of diverse learning models (learning objects) for the same subject, allowing for personalized learning paths tailored to each student's knowledge level, needs, and preferred learning styles. Learning sequences will serve as study organizers, structured around a cognitive skills taxonomy. While the environment primarily aims to support students with greater difficulties, it will also provide resources for advanced topics such as software engineering techniques, object orientation, and event programming, to promote the learning progress of all students.*

***Keywords:** programming learning, educational tools, personalized learning, collaborative tasks, learning environment, pedagogical approaches, software engineering, object-oriented programming, student engagement, adaptive learning paths, cognitive skills taxonomy, learning support.*

1. Introduction

Many higher education institutions offering computer science programs around the world face significant challenges with student failure in programming courses. Existing literature identifies several contributing factors, such as the required abstraction skills and problem-solving abilities necessary to learn programming effectively [1, 2]. Despite numerous proposed solutions, failure rates in these courses remain alarmingly high. It is recognized that while some students may naturally have a greater aptitude for programming tasks, all students have the potential to succeed if provided with the right guidance and are motivated to engage with the material.

Educators are aware of these challenges but often face limitations in offering personalized guidance due to the large class sizes that make individualized attention difficult. Each student has their unique background knowledge and learning pace, which often necessitates tailored instructional methods to ensure success.

Over time, various tools have been developed to assist in the teaching and learning of programming. Each of these tools offers distinct benefits. However, no single tool caters to the needs of every student, as the appropriate tool often depends on an individual's knowledge level and preferred study methods. Given the constraints of class sizes, it becomes nearly impossible for educators to provide the right tools at the right time for each student.

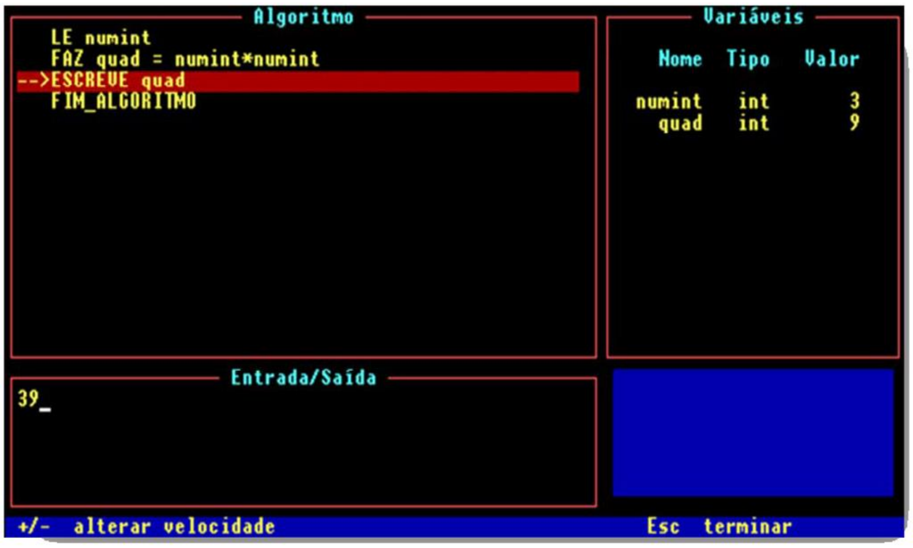
This paper outlines several tools developed by our research group and highlights the advantages they offer. Additionally, we present a new environment currently under development, which we believe will help address the challenges posed by class size limitations and the varying knowledge levels and learning paces of students. This environment is designed to make programming learning more accessible, adaptive, and effective, ensuring that students with diverse needs receive the appropriate support.

2. Developed Tools

2.1 VIP (Interactive Program Visualization)

VIP, which stands for *Visualização Interativa de Programas* (Interactive Program Visualization in Portuguese), was the first tool developed by our group in the late 1980s [3]. At that time, programming and the computing world were largely defined by the black screen with a blinking cursor, and tasks like debugging were particularly challenging. Explaining programming concepts to university students who had little or no prior experience with computers and programming was also a significant obstacle.

VIP was developed as a solution to these challenges. The system helped simplify the understanding of basic programming concepts by allowing students to write pseudo-code and visualize its execution. VIP enabled step-by-step simulation of pseudo-code, allowing students to observe the evolution of variable values and the program's output in real-time. This approach made learning programming more interactive and accessible, alleviating some of the difficulties associated with traditional text-based programming environments.



```
   LE numint
   FAZ quad = numint*numint
-->ESCREVE quad
   FIM_ALGORITMO
```

```
 Nome    Tipo   Valor

numint   int      3
  quad   int      9
```

```
39_
```

```
+/-  alterar velocidade                    Esc  terminar
```

Figure 1. The VIP interface.

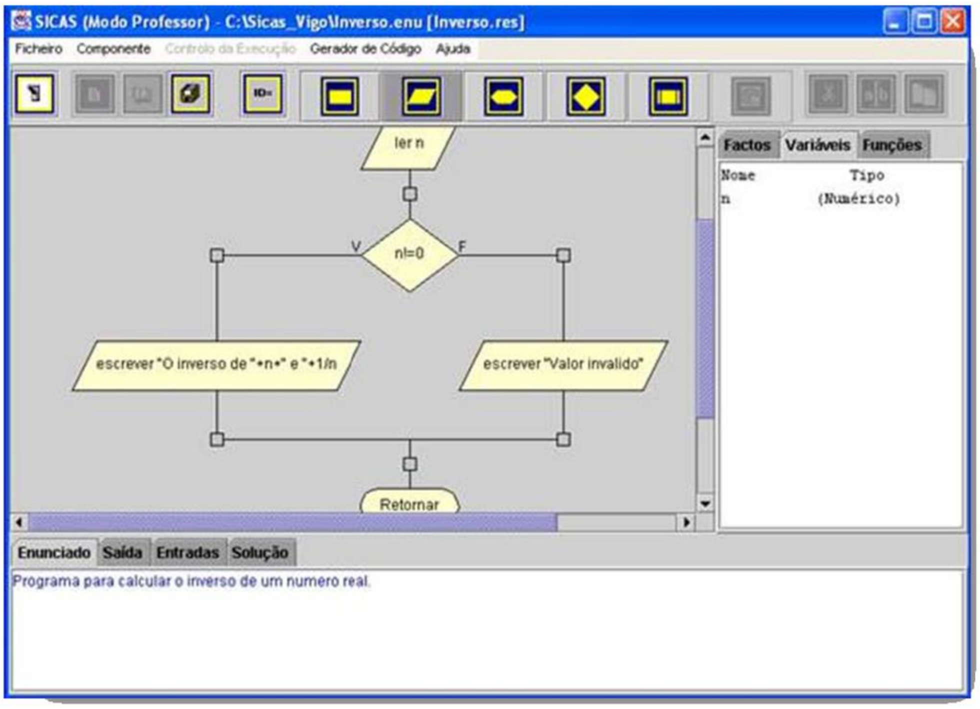
2.2. SICAS (2000)

In the late 1990s our group developed SICAS (a Portuguese acronym for Interactive System for Algorithm Development and Simulation, **Figure 2**). It was based on constructivist theories and aimed to create a pleasant programming tool that students may enjoy to use. SICAS doesn't include any explanatory material, but presents an environment that allows students to develop their capacities on the basis of practice, allowing them to design, observe, analyse and visually simulate algorithms. The idea is that students create solutions, detect eventual errors made, correct them and learn from those activities. This system allows students to implement algorithms to solve problems, using a flowchart representation. After that it is possible to execute/simulate the solution step-by-step, or continuously at a low or fast speed. The entire interface was constructed to be simple and easy to use. In this tool the complexities were minimized to focus the students' attention in the real task—problem solving.

SICAS provides support for the most common initial programming examples, since it allows the utilization of assignments, basic input/output, conditional structures and loops. It also permits the definition of functions to support more complex problems, including recursive functions. Although we tried to simplify the syntactical details students have to tackle, the environment has all the basic structures needed to start programming. It also supports common data types, namely numbers, strings and one-dimensional arrays.

Students can export their solutions to some other representations, namely Java, C and pseudo-code. Our representation options, allowing the students to write their algorithms without using a common programming language, stress our view that the important part is to develop algorithm creation skills, leaving the programming language syntactic details to a less important role.

SICAS has two working modes: a student mode and a teacher mode. The second has some extra features, allowing teachers to create new problems. This involves the problem description, a solution example and a set of input/output to allow the environment to test student solutions. A more complete description of SICAS can be found in [**4**,**5**].



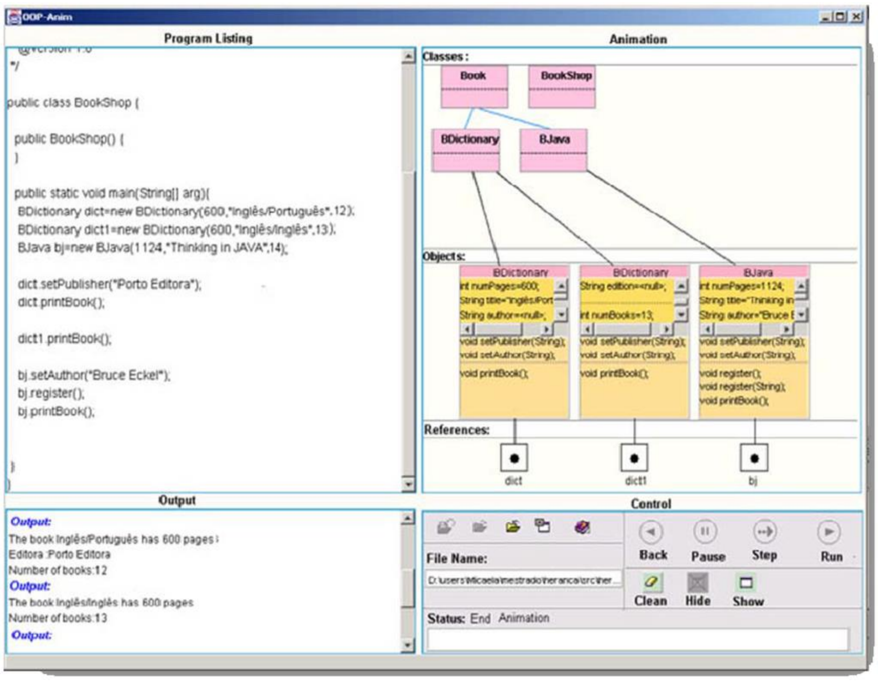
Programa para calcular o inverso de um numero real.

Figure 2. SICAS.

2.3. OOP-Anim (2004)

Based on SICAS' underlying ideas we also developed OOP-Anim (**Figure 3**). This system intends to help students understand basic object oriented concepts through the creation and visualization of programs. Like SICAS, OOP-Anim allows program simulation, but in this case they have to be written in Java. With OOP-Anim students can understand the effect of every instruction in terms of existing class definitions, object instances and related references.

Operations in OOP-Anim start with a Java program, normally written by the students. After that it is possible to control code execution and visualize the effect of each instruction in terms of program output and the object instances that will be created, accessed and modified. A more detailed description of OOP-Anim can be found in



```
*/

public class BookShop {

 public BookShop() {
 }

 public static void main(String[] arg){
  BDictionary dict=new BDictionary(600,"Inglês/Português",12);
  BDictionary dict1=new BDictionary(600,"Inglês/Inglês",13);
  BJava bj=new BJava(1124,"Thinking in JAVA",14);

  dict.setPublisher("Porto Editora");
  dict.printBook();

  dict1.printBook();

  bj.setAuthor("Bruce Eckel");
  bj.register();
  bj.printBook();


 }
}
```

```
Output:
The book Inglês/Português has 600 pages
Editora :Porto Editora
Number of books:12
Output:
The book Inglês/Inglês has 600 pages
Number of books:13
Output:
```

Figure 3. OOP-Anim.

2.4. SICAS-COL (2005)

Considering the potential of collaborative learning and educational simulation we decided to create a new version of SICAS that could support remote collaborative learning activities. The new tool was called SICAS-COL (**Figure 4**) [7] and resulted from the integration of SICAS with PlanEdit [**8**], a collaborative tool created within the scope of project Domosim-TPC [**9**], developed by our colleagues of the CHICO group at the University of Castilla—La Mancha (Spain).

PlanEdit was designed to support student discussions during problem solving group activities. It was first used in problem solving in the domotics field, but it can be used in other areas. By associating PlanEdit collaborative support with SICAS flowcharts, we were able to develop a powerful instrument to promote group discussion in the context of programming tasks. Each group member can propose solutions, expressed as SICAS flowcharts. With PlanEdit, students can contribute to a group solution. They can visualize proposals, discuss them, suggest modifications to proposed solutions, present alternative solutions, make comments, vote to express agreement or disagreement and choose the direction of teamwork development.

SICAS-COL is organized in three workspaces: a space for individual work, a space for group discussion and a space to share results. When the student is in the individual workspace, he has the opportunity to create, modify or visualize solutions. In the discussion workspace, students can discuss solutions and propose alternatives. The result workspace is used to store proposed solutions and other documents related with the problem at hand.

2.5. ProGuide (2005)

Although the use of SICAS has shown good results with some students [**10**], we verified that some other students needed a more guided approach, so in 2004, a new tool was developed, keeping in mind the difficulties of students that don't know how to begin to solve a problem. This system is ProGuide (**Figure 5**) and its main goal is to help reduce the difficulties that weaker programming students show in the initial learning stages [**11**]. It is based on a tutoring system model. ProGuide interacts with students when they are trying to solve a problem, using an algorithm representation similar to SICAS flowcharts. The system presents warnings about problems that can appear and questions students, trying to guide them in the different steps that lead to the problem solution. The interaction is based on internal structures that store information on problems and their solutions.

2.6. H-SICAS (2008)

One of the most important aspects that may influence learning success is the students' motivation. Any methodology, context or technology that can be a step forward in this process must be used. Technologies that student's are aware of and like can be an important mechanism to promote motivation. Therefore, they may help reach the desired success level in the learning process. Nowadays, students are familiar with most mobile devices, especially, computer science students. So, we decided to take advantage of the attractive features that these devices present, to have benefits in terms of an educational programming point of view.

H-SICAS (**Figure 6**) is an adaptation of SICAS to be used on mobile devices [**12**]. Perhaps the actual state of these devices is still not the best for programming learning, especially due to the limitations imposed by screen sizes and input interaction restrictions. However, as stated, the main goal is to exploit every kind of system or technology that can motivate students in the programming learning process.

3. Analysis of Tools

We think that each presented tool maintains its own identity and usefulness. However, none of the available tools completely fulfils the needs of all programming students. All these developments are justified because we have students with different levels. A very weak student could start with ProGuide. After the development of some abilities that already allow him/her to develop simple solutions, he/she could work individually using SICAS. It is also possible to initiate group work using SICAS-COL to support interactions between students, even at a distance. When students start to program using object oriented concepts, they can use OOP-Anim to help them to understand underlying concepts, object relations and dependencies.

If we consider a learning style model, such as the one proposed by Felder and Silverman [**13**], we can see that our tools tend to favour students with certain characteristics. One of the model dimensions characterizes students as verbal or visual. It is clear that our tools, making extensive use of visualization, are more oriented to visual students than to their verbal colleagues. We can find in the literature many references to studies that conclude that most engineering students are visual . However, we believe that a pedagogical environment should support different types of students. Ideally, the environment should adapt itself to each student learning style, presenting adequate learning activities. Thus, our proposal is centred firstly in a personalized education, which adapts the activities to each student in accordance with his/her cognitive state, rhythm and learning style.

We also noticed that flowcharts are relatively easy to understand by students. In general, students understand examples made with SICAS and are able to predict their output. Moreover, after analyzing some examples, many of them are able to solve simple problems. Using the SICAS options for code generation, students understand that the conversion from flowcharts to a real program is almost direct However, when we ask students to close SICAS and create programs from scratch, in a classical environment, the difficulties start again. Initially, SICAS was planned only for initiating students in very simple programming tasks. Nevertheless, the experience has shown that it is desirable to extend the period of its use. The tool should help in the next learning phase when students start to code directly in a programming language. For that, we must extend SICAS features not only in terms of the language itself, but also to include other concepts, making SICAS's lifecycle and utility longer. We must extend data types, support object oriented programming and other common programming activities. Support to remote teamwork and synchronized mechanisms that allow program development using flowcharts or real code should also be included. However, even extending the environment characteristics, we don't want to lose one of the best features of SICAS, the possibility that students have to create and simulate their own algorithms/programs.

4. Proposal

Each of the tools presented in the previous section was planned for a specific context, subject, a limited set of lessons or to help in workgroup projects. Every tool has a major or minor contribution but also an additional workload in the learning process due to the time needed to understand each of them. Every time we want to present a new tool, even simple and user friendly, we need to explain some fundamental aspects of its interface and functionalities. Sometimes, the time spent exceeds the time that the activity needs. Associated with this, teachers are frequently confronted with heterogeneous classes with different problems, occurring during tool presentation, which compromises the normal class evolution.

The heterogeneity issue represents a serious problem because it is something that the teachers are confronted with right from the beginning of the semester. Each student has his/her own knowledge level, learning style, goals and social context. Each student has his/her own pace!

In the first few lessons, when the teacher starts to introduce the concepts, the feedback that she/he usually receives comes from the students that already have some background in the field. Other students that have some difficulties postpone their doubts, hoping that their doubts will be clarified with the rest of the class. As a result, teachers follow the more advanced students' pace. When the teacher realizes the real situation she/he is forced to reorganize activities, concentrating her/his efforts on students that show more problems. Sometimes, it's like starting from scratch. This duality creates many problems to the instructor, as it is very difficult to find a rhythm and level of detail that is adequate to most students.

Another point that we must be aware of is the fact that programming learning is, essentially and understandably, based on practical activities. Nevertheless, we can't forget the importance of lectures about concepts and other activities usual in programming courses. It would be beneficial to join diverse tools and teaching approaches that create a consistent environment. This environment should permit a balanced skill development, both theoretical and practical and adapted to each student pace.

Nowadays, one of the more used methods to grant access to contents is the e-Learning platform. It has the advantage of allowing students to access contents asynchronously, at their own rhythm. However, the usual e-Learning platforms are not prepared for programming particularities.

What we are now proposing is to take advantage of an e-Learning platform, already known by students, enriched with a set of tools adapted to programming learning needs. One of the most important features is the ability to define learning paths tailored to each student's actual level, eventually based on student learning styles and the student evolution. These learning paths are formed by lessons on fundamental concepts, but also include exercises that permit knowledge consolidation. Students must be able to access the right tools that are adapted to each exercise, which should be chosen automatically by the e-Learning platform. It is also important that the platform and tools allow collaboration among students and between teacher and students.

The prototype that we are now developing is integrated in the Moodle e-Learning platform. We are developing some new types of activities to include in Moodle that can answer the needs already identified. It is essential for the definition of learning paths to introduce mechanisms that can restrict access to some activities. In the next version of Moodle, version 2.00, it is already expected that some of these mechanisms will be implemented, such as the "Conditional activities" [**25**]. Nevertheless, we think that this new improvement is not enough for our purposes. Different students should have different conditions to trigger each activity. It's also expected that the next version will implement "Progress tracking" which will permit the definition of personal learning plans based on courses completed or outcomes reached. Even if these new features are available in the next version, we need to enrich them with some automation to help teachers define different and appropriated plans for each student. To automate the process we will take into account the results of some inquiries (to test learning styles and programming background), the competences associated with each activity, the skills that students should develop and, when possible, successful and unsuccessful history cases.

Another essential tool that should be worked on is related with assignments development. In the present version of Moodle, there are some types of assignments, but they only permit the organization and control of the delivery process. Moreover, the current available assignment types were not thought for programming contexts. Even with the more advanced type of assignment, that permits multiple file submission, we can't create a repository that allows effective teamwork. When one uploads a new file, other members can't access it. We have already developed a new assignment prototype (**Figure 7**) that allows the constitution of a group portfolio by all workgroup elements. This new type of assignment has associated a system that permits the edition of each file, simultaneously and synchronically, by all group elements. This system provides a secure access to Moodle platform files, available for that group in one precise subject/course. For now documents are always assumed as text files but we want to extend the features of this system. For that, we are already developing a new SICAS (SICAS NG), based on the original one, but enriched with features that we identified as essential during these past years.

SICAS NG will follow the original goal of SICAS: being an easy to use tool that allows the creation and simulation of algorithms using a flowchart representation. Moreover, we also want the new version to be an effective tool for more advanced stages in the programming learning process. In this new version, the object concept will be supported and related concepts like encapsulation, inheritance and polymorphism, will be available. Other types of representations will be used alongside with flowcharts, namely UML and text based representations. Another feature of this new version is the possibility to collaborate in a programming project. This feature should allow simultaneous editing by the group elements. It is also planned that in SICAS NG students can ask for their teachers help. If the teacher is online, it will be possible for her/him to integrate a collaborative session, make changes to the code or comment on some parts. SICAS NG can also be used in a classroom environment to let the teacher build examples, step-by-step, collaboratively with students. Students can follow the construction of the example on their own computer, in the classroom or some other place. During the explanation, the teacher can ask a student or the class to make a specific improvement of an example. After that, the teacher can present a personal solution or choose one of the students' solutions as a good (or bad) example to the class. All the SICAS NG projects can be available through Moodle platform or stored locally.

5. Conclusions

Over the years, numerous efforts have been made to enhance the learning of programming and provide support to students encountering difficulties in this area. Despite these efforts, the results remain inconsistent, with clear and widespread success yet to be achieved. One of the primary issues is the inherent difficulty of the programming learning process itself, which often leads to high failure rates. These difficulties are compounded by the large class sizes typically seen in computer science education, where the teacher must address the needs of an enormous number of students.

Another significant challenge is the heterogeneity within these classes, where students possess varying levels of knowledge, skills, and learning paces. Some students may have a strong background in programming, while others are just beginning to grasp basic concepts. In such a setting, traditional, one-size-fits-all teaching methods struggle to provide effective support. As a result, it becomes exceedingly difficult for teachers to give each student the attention they need to succeed, often leading to frustration and disengagement from the learning process. This classical approach, constrained by time, resources, and the diversity of student needs, is inevitably prone to failure.

In response to these challenges, our research group has undertaken various initiatives to improve programming education. These efforts have led to the development of a suite of tools aimed at supporting both teaching and learning activities related to programming. Each tool was designed with a particular aspect of the learning process in mind, focusing on increasing interactivity, providing visual feedback, and catering to students' diverse cognitive needs.

We have also been working on the development of a new, more comprehensive learning environment that aims to address the limitations of current approaches. This new environment integrates various technologies, providing a multi-faceted platform that can support a broad range of learning activities. By offering personalized learning experiences, this environment aims to accommodate different student cognitive needs, allowing for tailored support that adapts to each learner's individual pace and understanding.

This platform is designed not only to assist students who struggle with the learning process but also to provide advanced learners with resources to further enhance their understanding. Through the integration of diverse pedagogical approaches, learning tools, and technologies, we envision an environment that can foster greater engagement, enhance comprehension, and ultimately reduce failure rates in programming courses. By making learning more interactive, personalized, and supportive, this environment strives to create a more inclusive and effective programming education experience for all students.

References

1. Bennedsen, J.; Caspersen, M. Abstraction ability as an indicator of success for learning object-oriented programming? *SIGCSE Bullet.* 2005, *38*, 39–43. [Google Scholar] [CrossRef]
2. Jenkins, T. On the difficulty of learning to program. In Proceedings of the 3rd Annual LTSN_ICS Conference, Loughborough, UK, August 2002; pp. 53–58.
3. Mendes, A.; Mendes, T. VIP—A tool to visualize programming examples. In Proceedings of the EACT 88—Education and Application of Computer Technology, Malta, October 1988.
4. Gomes, A.; Mendes, A.J. SICAS: Interactive system for algorithm development and simulation. In *Computers and Education: Towards an Interconnected Society*; Ortega, M., Bravo, J., Eds.; Kluwer Academic Publishers: Dordrecht, The Netherlands, 2001; pp. 159–166. [Google Scholar]
5. Gomes, A. Ambiente de suporte à aprendizagem de conceitos básicos de programação. MSc Thesis, Faculdade de Ciências e Tecnologia da Universidade de Coimbra, Coimbra, Portugal, November 2000. [Google Scholar]
6. Esteves, M.; Mendes, A.J. OOP-Anim, a system to support learning of basic object oriented programming concepts. In Proceedings of the CompSysTech'2003—International Conference on Computer Systems and Technologies, Sofia, Bulgaria; 2003. [Google Scholar]
7. Rebelo, B.; Mendes, A.; Marcelino, M.; Redondo, M. Sistema Colaborativo de Suporte à Aprendizagem em Grupo da Programação—SICAS-COL. In Proceedings of the VII Simpósio Internacional de Informática Educativa, Leiria, Portugal, November 2005.
8. Redondo, M.A.; Bravo, C.; Ortega, M.; Verdejo, M.F. PlanEdit: An adaptive tool for design learning by problem solving. In Proceedings of the 2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH2002), Malaga, Spain, May 2002; pp. 560–563.
9. Redondo, M.A.; Bravo, C.; Bravo, J.; Ortega, M. Organizing activities of problem based collaborative learning with the DomoSim-TPC system. In *Computers and Education: Towards a Lifelong Learning Society*; Llamas, M., Fernández, M.J., Anido, L.E., Eds.; Kluwer Academic Publishers: Dordrecht, The Netherlands, 2003; pp. 37–49. [Google Scholar]
10. Gomes, A.; Mendes, A.; Marcelino, M. Avaliação e evolução de um sistema de apoio à aprendizagem da programação. In Proceedings of the VII Congresso Iberoamericano de Informática Educativa, Monterrey, México, October 2004.

11. Areias, C.M.; Mendes, A. A dialogue-based tool to support initial programming learning. In Proceedings of the 3rd E-Learning Conference—Computer Science Education, Coimbra, Portugal, September 2006.

12. Marcelino, M.; Mihaylov, T.; Mendes, A. H-SICAS, Handheld algorithm animation and simulation tool to support initial programming learning. In Proceedings of the 38th ASEE/IEEE Frontiers in Education Conference, New York, NY, USA, October 2008.

13. Felder, R.M. Learning and teaching styles in engineering education. *J. Eng. Educ.* 1988, *78*, 674–681. [Google Scholar]

14. Rosati, P.A. Comparisons of learning preferences in an engineering program. In Proceedings of the 26th Frontiers in Education Conference, Salt Lake City, UT, USA, November 1996.

15. Constant, K.P. Using multimedia techniques to address diverse learning styles in materials education. *J. Mater. Educ.* 1997, *19*, 1–8. [Google Scholar]

16. Paterson, K.G. Student perceptions of internet-based learning tools in environmental engineering education. *J. Eng. Educ.* 1999, *88*, 295–304. [Google Scholar] [CrossRef]

17. Rosati, P.A. Specific differences and similarities in the learning preferences of engineering students. In Proceedings of the 29th Frontiers in Education Conference, San Juan, Puerto Rico, November 1999.

18. Buxeda, R.; Jimenez, L.; Morell, L. Transforming an engineering course to enhance student learning. In Proceedings of the ICEE 2001 International Conference on Engineering Education, Oslo/Bergen, Norway, August 2001.

19. De Vita, G. Learning styles, culture and inclusive instruction in the multicultural classroom: A business and management perspective. *Innov. Educ. Teach. Int.* 2001, *38*, 165–174. [Google Scholar] [CrossRef]

20. Livesay, G.A.; Dee, K.C.; Nauman, E.A.; Hites, L.S.J. Engineering student learning styles: A statistical analysis using felder's index of learning styles. In Proceedings of the 2002 ASEE Conference and Exposition, Montreal, QB, Canada, June 2002.

21. Lopes, W.M. ILS-Inventário de Estilos de Aprendizagem de Felder-Soloman: Investigação de sua Validade em Estudantes Universitários de Belo Horizonte. MSc Thesis, Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brazil, 2002. [Google Scholar]

22. Kuri, N.P.; Truzzi, O.M. Learning styles of freshmen engineering students. In Proceedings of the 2002 International Conference on Engineering Education, Manchester, UK, August 2002.

23. Seery, N.; Gaughran, W.F.; Waldmann, T. Multi-modal learning in engineering education. In Proceedings of the 2003 ASEE Conference and Exposition, Nashville, TN, USA, June 2003.

24. Zywno, M.S. A contribution of validation of score meaning for Felder-Soloman's index of learning styles. In Proceedings of the 2003 ASEE Conference and Exposition, Nashville, TN, USA, June 2003.

25. Moodle 2.0 Roadmap. Available online: http://docs.moodle.org/en/Roadmap (accessed on 25 February 2010).