Programming Assistance Software Tools to Support the Teaching of Introductory Programming

Melisa Koorsse
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 372 2193
Melisa.Koorsse@nmmu.ac.za

André P. Calitz
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 504 2639
Andre.Calitz@nmmu.ac.za

Charmain B. Cilliers
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 504 2235
Charmain.Cilliers@nmmu.ac.za

ABSTRACT

Novice programmers find learning to program difficult and debugging has also been identified as a difficult task for novice programmers. Novice programmers struggle to develop accurate mental models of programming concepts and processes, have difficulty understanding how a computer executes instructions and struggle to apply the syntax rules of high-level programming languages. Different programming assistance software tools have been developed to assist novice programmers with their understanding of programming concepts. Programming assistance tools use different techniques to assist novice programmers, including visualisation and animation techniques, and drag and drop interfaces. A number of programming assistance tools has shortcomings, for example, not supporting all introductory programming concepts.

This paper identifies several different programming assistance software tools that are freely available for use by novice programmers. The programming assistance tools are evaluated using selection criteria that can be used to select programming assistance software tools for use in introductory programming courses. The selection criteria are formulated from a literature review of introductory programming as well as research conducted with Information Technology (IT) learners in South African secondary schools. The research presented in this paper aims to provide IT teachers and introductory programming lecturers with a list of programming assistance software tools that are freely available for introductory programming courses and subjects, selection criteria that can be used to evaluate the programming assistance tools and a discussion of some of the shortcomings of programming assistance tools that need to be considered when selecting tools for introductory programming courses.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer Science education, Curriculum.*

General Terms

Performance, Human Factors, Languages.

Keywords

Introductory Programming, Novice Programmers, Information Technology, Programming Assistance Tools.

1. INTRODUCTION

Expert or professional programmers possess problem solving abilities [1, 2] that are essential for developing software that is fast and scalable. These abilities are developed from programming experience gained over an average of 10 years [30].

Novice programmers, including Information Technology learners at South African secondary schools, find learning to program to be a difficult task [32]. The reason for this is that novice programmers need to learn how to understand and solve a problem, formulate a solution in a structured form (algorithm) and then write the algorithm in a specific programming language [34]. Programming can be a difficult task if programmers are unable to plan solutions [31], lack understanding of programming concepts due to the abstract nature of these concepts [23] and lack understanding of how a computer executes code [5].

The teaching and learning of programming concepts can be supported with programming assistance tools. Research in novice programming has suggested and developed programming tools [26] to enhance comprehension of algorithms and computer programs, assist with code debugging and assess programming knowledge and skills. The programming assistance tools use different techniques such as visualisation, animation or drag-and-drop interfaces to improve the conceptual understanding of programming concepts [2].

Educators and students may be unaware of the different programming assistance tools that can be used to support understanding of programming concepts. Certain tools have educational deficiencies and do not support all of the content presented in an introductory programming course.

This paper discusses the difficulties novice programmers experience when learning to program (Section 2). Criteria that can be used to select programming assistance tools to support novice programmers are formulated (Section 3) and used to evaluate several programming assistance tools freely available for use by novice programmers (Sections 4 and 5). The paper is concluded and future work is presented in Section 6.

2. PROGRAMMING DIFFICULTIES

Programming is a complex activity that requires a novice to learn non-trivial facts, skills and concepts that are new to them [2].

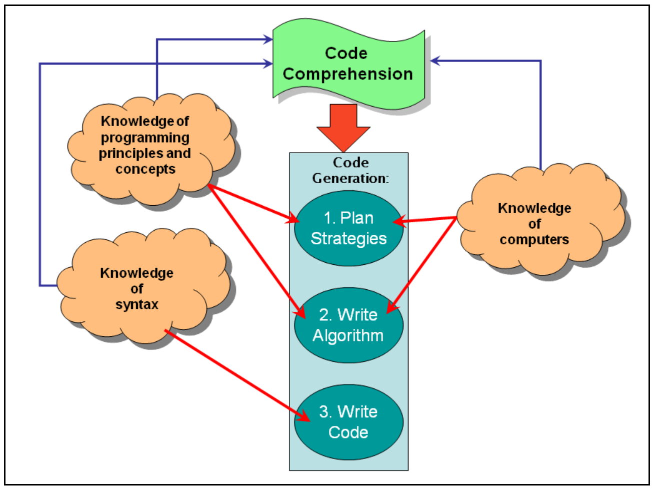


Figure 1. Knowledge and skills required by programmers.

Figure 1 summarises the relationship between code comprehension and generation and the different types of knowledge a novice programmer requires.

Code generation involves 3 main steps (Figure 1):

1. A given problem statement or requirements set must be considered in order to decide upon the programming strategy to use.
2. An algorithm to solve the problem should be formulated, often using pseudocode.
3. The algorithm is translated into the code of the programming language being used. The program is tested and changed as necessary until the original set of requirements to solve the problem, are met.

The three steps outlined above can only be achieved by a programmer who is able to apply programming knowledge and strategies and who has the ability to comprehend as well as generate code [8, 30].

2.1 Programming Knowledge

Programming knowledge includes knowledge of programming concepts and principles, knowledge of computers and knowledge of programming language syntax (Figure 1) [26]. Knowledge of programming concepts and principles is an understanding of how different concepts are implemented and why. For example, how a *for*-loop works or the purpose of a variable. Knowledge of computers includes an understanding of how computer events occur and can be handled by the code (for example a mouse-click or button press). Knowledge of syntax is required in order to implement a solution in a particular programming language.

All three of the above knowledge areas are important for code generation. If a programmer is unaware of the different programming concepts it would be difficult to plan a suitable solution. A well designed solution will not run successfully if the syntax used is incorrect. Code not executed in the correct order or when certain events occur will result in an incorrect program solution, regardless of whether there are no syntax errors or if the correct programming concepts have been implemented.

2.2 Programming Strategies

A programming strategy is the way in which programming knowledge is applied to solve a particular problem [8, 30]. A suitable programming strategy is required for the first step of code generation (Figure 1). A programmer who has an understanding of programming knowledge, but who is unable to use their knowledge to solve or transfer solutions from simple problems to more complex problems, lacks an understanding of programming strategies [8, 23]. Similarly, a programmer lacks programming strategy or problem solving ability if the programs compile and run yet do not solve the problem due to logic errors [1].

In general, more time is spent teaching programming language knowledge than programming strategy [1, 26]. Novice programmers tend to combine the steps of the code generation process (Figure 1) as they attempt to write the algorithm in a particular programming language [11, 31].

2.3 Code Comprehension and Generation

A novice programmer should have adequate knowledge in all three programming knowledge areas (Section 2.1) to be able to comprehend and generate code successfully (Figure 1). A novice programmer that lacks ability in one or more of the knowledge areas will struggle to generate a code solution.

Program comprehension is described as a "*cognitively complex skill*" [3, 24]. Novice programmers reading code should be able to identify the knowledge such as concepts used in the solution and the strategy applied to produce the solution [8].

A correlation has been shown to exist between code comprehension and code generation [8, 25]. Code comprehension is regarded as an important skill required for successful programming [3]. Novice programmers that are not able to read and understand code are unable to write similar code [25]. This can be a problem as new concepts are explained to novice programmers using practical examples. If a novice programmer is unable to read and understand code solutions, they will be unable to build their knowledge of programming concepts and strategies to solve real-world problems [24, 25]. The ability to read and understand code is moreover an important skill required for finding logical errors in code [25].

2.4 Other Contributing Factors

Other factors that can contribute to make learning to program difficult include the teaching approach used, the programming language and environment and specific programming concepts that are difficult to understand.

2.4.1 Teaching Approach

A novice programming student is guided by the teacher presenting the programming course when learning to program. Teachers need to present the course with a balance between the "how to" and "why" explanations. Overemphasising "how to", for example, how to use an *if*-statement in a particular problem, may result in students being unable to transfer what they have learnt. Their underlying skills and concept knowledge would be lacking [2]. Overemphasising "why" would provide students with a theoretical knowledge of the underlying programming principles. However, the theory would need to be accompanied by practical experience demonstrating how the principles are applied.

Teachers also need to assist novice programmers to create a proper mental model of different programming concepts, especially the more abstract concepts [32]. Each person has a preferred learning style, differing abilities, learning speeds and attitudes or motivations toward the subject [22, 23] which would need to be taken into consideration by the teacher when assisting individual students. However, many teachers use one teaching approach for all students [1] thus not catering for the learning requirements of individual students.

2.4.2 Programming Language and Environment

The programming language and development environment tool used by novice programmers when learning to program can also contribute to the difficulties experienced [26]. Certain programming languages are too complex to use to explain or teach introductory programming concepts [5, 17, 27]. If a novice programmer is having difficulty understanding a programming concept, an explanation of the concept using a code example should not be further confusing. Professional programming environments may also overwhelm novice programmers by presenting them with functionality and interfaces that are only used by professional programmers [5, 17, 27].

2.4.3 Specific Programming Concepts

Certain programming concepts are more abstract than others and are thus more difficult for novice programmers to understand [16]. Abstract concepts have no related explanation in real life, making it difficult for novice programmers to develop an accurate mental model of these concepts.

A literature survey has identified several specific programming concepts which novice programmers struggle to understand. *Recursion* is identified by three different research studies [14, 23, 32] as a difficult concept for novice programmers to understand. *Abstract data types* such as *arrays* are identified by two separate research studies [14, 23]. Novice programmers also seem to have difficulty understanding and using *methods* or *procedures and functions* [12, 14].

Another concept that teachers specifically find difficult to teach is object-oriented programming (OOP). The difficulty related to OOP has been associated with the paradigm shift from structured methods and not the actual concepts [17]. Two approaches have been recommended for teaching OOP: "objects first" and "objects last". The "objects last" approach is the most common instruction method. This approach starts with simple concepts and programs and gradually advances to more difficult concepts [7]. This provides a gentle learning curve which allows novice programmers to incrementally build their programming knowledge.

A problem with teaching OOP last is that a paradigm shift is needed for students to switch from the procedural style of programming to the OOP style of programming. This shift has been identified as the cause of the difficulties related to teaching OOP last [7, 17]. The solution is the "objects first" approach. The "objects first" approach introduces concepts such as string handling and looping within the OOP environment. However, a novice programmer's first experience of programming is a difficult mental challenge because they are faced with learning the basic programming concepts and syntax together with the complexities of OOP [7].

A review of the literature has indicated greater support for the "objects first" approach [7, 17, 18, 20]. This is evident from many tools that have been developed to promote "objects first" and ensure that students are not impacted by the difficulties of "objects first" highlighted earlier.

3. SELECTION CRITERIA

Table 1 lists selection criteria that can be used to select a programming assistance tool to support novice programmers learning to program. All items followed by an asterisk are derived from the programming difficulties identified in Section 2. The remaining items are derived from the results of surveys administered to Information Technology (IT) learners and teachers in South African secondary schools [21, 22].

The left hand column (Table 1) lists programming concepts that an introductory programming course should include. This list is derived from the list of programming concepts that Information Technology (IT) learners in South African secondary schools are required to learn [9]. The bold items in the list are programming concepts that have been identified as difficult to understand based on the results of a survey administered to IT teachers and IT learners [21].

The programming skills and knowledge items in the right hand column of Table 1, originate from the programming skills required by novice programmers identified in the literature survey presented in Section 2. Programming assistance tools should allow novice programmers to develop code comprehension skills, promote problem solving ability using appropriate strategies, assist understanding of code execution and allow syntax knowledge and knowledge of programming principles and concepts to be improved.

Every person has a preferred learning preference. Four learning preferences are identified by Fleming and Baume [10], namely visual, aural, read/write and kinesthetic. A persons learning preference could be only one or a combination of the four. The IT teacher/learner survey results indicated that programming assistance tools should cater for at least 2 of the learning preferences to assist individual users.

A programming assistance tool should be constructivist to allow a novice programmer to "build" their knowledge of programming concepts using the tool and promote self-study. Novice programmers also need assistance to formulate a code solution before actually writing the program code.

A tool that uses visualisation and/or animation techniques can increase interest and motivation [29]. Error handling refers to whether or not a tool can detect errors in code, that is, compile a code solution. Simple errors messages and/or suggestions to correct errors refer to the way in which the tool assists novice programmers to detect and correct errors in the code.

Novice programmers also struggle to apply programming strategies used to solve simple exercises, to solve more complex exercises.

Table 1. Selection criteria for Programming Assistance Tools.

Selection Criteria	
Concepts	***Programming skills & knowledge:***
Variables	Code comprehension*
Input (getting information from the user)	Promotes problem solving using strategies*
Output(displaying information to the user)	Code execution*
If-statements	Syntax knowledge*
Switch statements	Knowledge of programming principles & concepts*
For-loops	***Teaching/Learning approach:***
Repeat-until/do-while loops	Constructivist (promote self-study)
While-do loops	Feedback to guide solution creation
String handling	***Learning Preferences:***
Procedures*	Visual
Functions*	Aural
One-Dimensional Arrays*	Read/Write
Two-Dimensional Arrays*	Kinesthetic
File handling	***Other:***
Accessing a database	Simple & complex examples (scaffolding)
SQL statements	Error handling
Correct use of parameters	Simple error messages/suggestions to correct errors
Objects & classes*	Visualisation/Animation
Problem solving*	***Programming Language (e.g. Java, C#, Delphi (Pascal), etc.)***
Debugging*	

The programming assistance tools should be able to assist novice programmers with the syntax of whichever programming language the novice programmer is learning to program in. Care should be taken to select a tool which implements code or can be adapted to implement code that is the same or similar to that of the programming language being used by the novice programmers. Differences in syntax or the manner in which concepts are implemented can be confusing and result in making the learning process more difficult.

4. PROGRAMMING ASSISTANCE TOOLS

Programming assistance tools (PATs) are specifically designed to support novice programmers learning to program [26]. PATs can assist novice programmers to develop their understanding of programming concepts. [31] states that the ideal PAT would be able to support several features including problem solving, algorithm design, assist with learning syntax for a particular programming language, and partial compiling to quickly check output and operation of a code block.

PATs can also make use of visualisation and animation techniques. Most programming concepts, data structures and algorithms are abstract [31]. Visualisation techniques can be used to help novice programmers develop an accurate mental model of programming concepts (Section 2).

Several PATs have been identified from literature and will be discussed briefly.

4.1 RobotProg

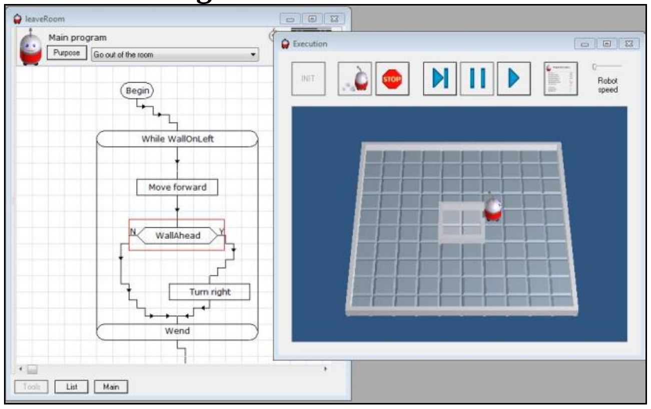


Figure 2. Execution of RobotProg flowchart.

RobotProg is a PAT in which the user creates a flowchart (Figure 2) by dragging and dropping icons representing programming concepts. When the program created is executed, it controls a robot to perform specific tasks (Figure 2).

Different levels of difficulty can be specified in RobotProg. The simple programming concepts are available in the lowest level. More complex programming concepts are available for use in the flowchart as the level is increased.

Users are also challenged to complete tasks such as finding a corner or picking up a ball. The RobotProg tool is able to detect whether or not the task has been completed successfully. The RobotProg interface can be complicated for novice programmers to understand in the beginning. Users are not able to view any code generated by the flowcharts.

4.2 BlueJ

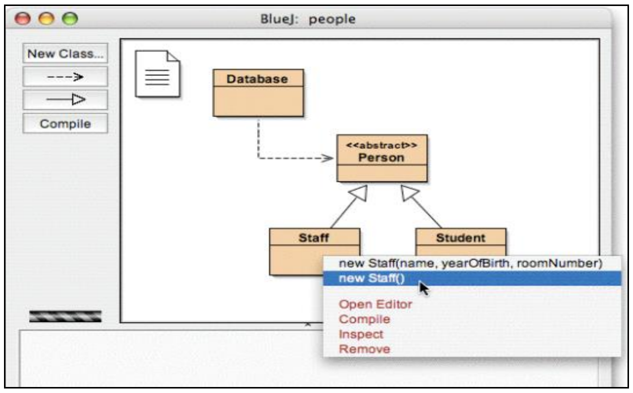


Figure 3. Creating UML-like class diagrams in BlueJ.

BlueJ is a tool that uses an objects first approach (Section 2.4) to introduce novice programmers to the concept of objects and classes. In BlueJ, the objects and classes concepts are demonstrated without the user having to write any code [17, 19].

The advantages of BlueJ are that it is interactive and simple to use. It uses visualisation to help novice programmers understand objects and classes. UML-like class diagrams are used to provide a graphical overview of project structures (Figure 3). A disadvantage is that exercises would need to be designed by the teacher, based on the functionality provided by BlueJ.

BlueJ generates code in Java. If users want to view the corresponding code implementations associated with the objects and classes visualisations, an understanding of the Java programming language is required. BlueJ also does not provide assistance with the understanding of other programming concepts such as conditional statements or looping, although they can be implemented.

4.3 Greenfoot

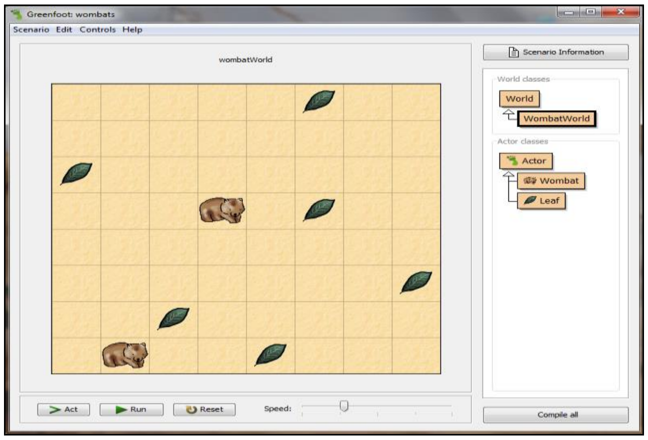
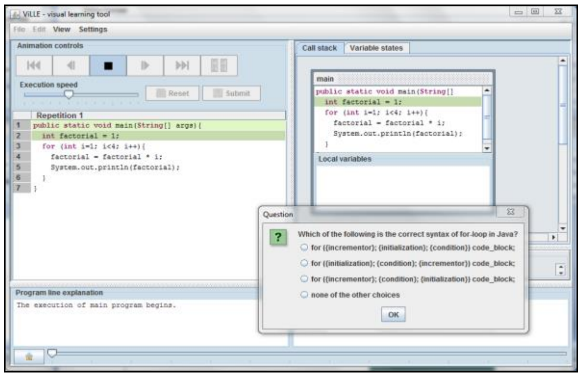


Figure 4. Greenfoot main screen with objects.

Greenfoot is a PAT that is used to teach object-oriented programming to secondary school learners [15]. Users can easily create different microworlds that are visually appealing and easy to interact with.

Users can interact with Greenfoot objects directly (Figure 4). Changes in the position and appearance of objects can be observed directly. Classes associated with Greenfoot objects are visible on the main screen (Figure 4) and code for the different objects can also be modified by the user. The coding language used is Java. Similar to BlueJ, Greenfoot only assists with the understanding of the implementation of objects and classes.

4.4 Ville



```
public static void main(String[] args){
  int factorial = 1;
  for (int i=1; i<4; i++){
    factorial = factorial * i;
    System.out.println(factorial);
  }
}
```

Which of the following is the correct syntax of for-loop in Java?
- for ({incrementor}; {initialization}; {condition}) code_block;
- for ({initialization}; {condition}; {incrementor}) code_block;
- for ({incrementor}; {condition}; {initialization}) code_block;
- none of the other choices

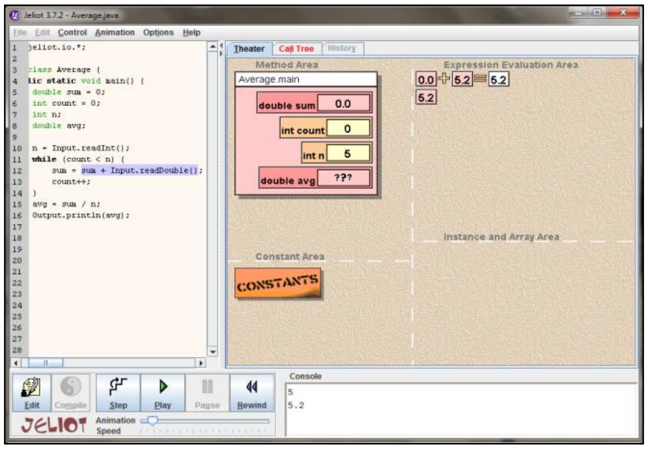
The execution of main program begins.

Figure 5. Execution of program in Ville with question being posed to user.

Ville is a language-independent programming tool [28]. Code execution is demonstrated using visualisation techniques. Ville has the syntax rules for several programming languages built in, including Java, Python, PHP, javascript, C++ and pseudocode. New languages can be added using the syntax editor.

A user can control the speed of execution as well as step forward or back through the program code. Explanations for program lines are provided and Ville can be set up to ask the user questions about the current code being executed (Figure 5).

4.5 Jeliot



```
jeliot.io.*;

class Average {
lic static void main() {
 double sum = 0;
 int count = 0;
 int n;
 double avg;

 n = Input.readInt();
 while (count < n) {
     sum = sum + Input.readDouble();
     count++;
 }
 avg = sum / n;
 Output.println(avg);
```

```
5
5.2
```

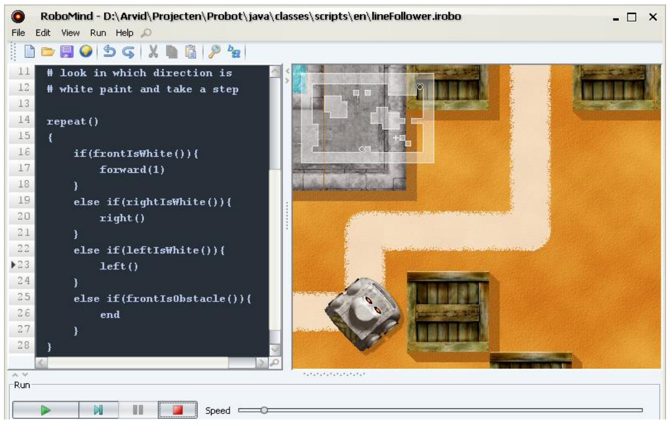
Figure 6. Visualisation of program execution using Jeliot.

Jeliot animates programs to assist novice programmer understanding of introductory programming concepts. Jeliot is capable of animating most of the Java language, including object allocation [4].

Visualisation and animation techniques are used in Jeliot to assist novice programmers to develop an accurate mental model of programming concepts during code comprehension [4]. The current line of execution is indicated to users during execution of the program (Figure 6). Four areas in the animation are used to indicate current variable values, expression evaluations, value of constants and the allocation of and reference to objects and arrays.

The standard Jeliot program assists novice programmers with the understanding of Java programs [4]. Jeliot 3 has been redesigned to separate the interpretation and animation of Java programs. This means that Jeliot 3 can be used to animate programs written in another programming language.

4.6 RoboMind


RoboMind - D:\Arvid\Projecten\Probot\java\classes\scripts\en\lineFollower.irobo
File
Edit
View
Run
Help
Run
Speed

```
# look in which direction is
# white paint and take a step

repeat()
{
    if(frontIsWhite()){
        forward(1)
    }
    else if(rightIsWhite()){
        right()
    }
    else if(leftIsWhite()){
        left()
    }
    else if(frontIsObstacle()){
        end
    }
}
```

Figure 7. RoboMind program executing.

RoboMind has been designed as a tool that can be used as a first introduction to programming without any prerequisites. User program a robot to move around and interact with objects in a map world using a simple educational programming language called ROBO (Figure 7).

The RoboMind environment is freely available and the RoboMind 2.2 development environment is available as open source. RoboMind can thus be adapted to change the implementation of programming concepts to suit a particular programming language or to add additional functionality.

4.7 Scratch

The purpose of Scratch is to provide children with a tool that will allow them to start programming earlier [33]. Scratch allows people of different backgrounds and interests to easily create their own interactive games, animations, stories and simulations [29, 33].

In Scratch a building block metaphor is used whereby graphical blocks are combined to build scripts (Figure 8). This allows novice programmers to focus on finding problem solutions as syntax errors are eliminated. Scratch is also visually appealing and promotes active learning.

A problem that novice programmers using Scratch may encounter is that it will be difficult for them to move directly to a traditional programming environment after using Scratch. The use of an intermediate software tool to provide a link between the concepts introduced in Scratch and the methods of implementing these concepts in a programming language is suggested [29].

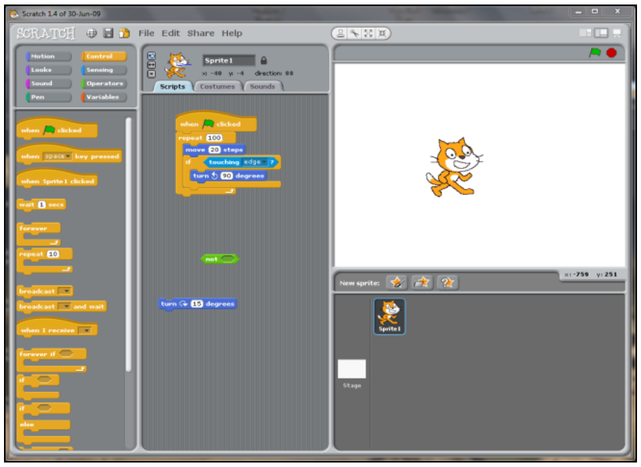


Figure 8. Scratch main screen with code area in the middle.

4.8 Additional Programming Assistance Tools

Several other PATs were also identified by this research study. PlanAni [6], for example, is a tool that uses animation to demonstrate the roles of variables. Alice [7, 15] is a tool similar to Scratch that can be used to create 3D animations and games by dragging object properties and methods to build the program code. In B# [13], users create a program by dragging and dropping icons to create a flowchart. The program can be executed and equivalent Pascal code is generated. jGrasp [17] automatically generates UML class diagrams to allow users to visualise objects, data structures and primitive variables.

5. EVALUATION OF PATS

This section demonstrates how the selection criteria formulated in Section 3 can be used to evaluate programming assistance tools. The programming assistance tools presented in Section 4 are evaluated. Table 2 and Table 3 provide an indication of which selection criteria each of the tools meet.

BlueJ, Greenfoot, BlueJ and Jeliot allow users to write programs in Java code, while B# generates Pascal code from the flowchart created by the user. The remaining tools are not programming language specific. Scratch, Alice and RobotProg can be used to teach any programming languages even though none of the tools explicitly teaches syntax for these languages. All three tools allow users to learn about different programming concepts using a drag-and-drop interface. The statements used are similar to the statements used by most programming languages even though they do not conform to the syntactical rules of any particular language. RoboMind can be adapted to compile code in any programming language.

BlueJ, Greenfoot, jGrasp and Jeliot allow the user to implement all of the programming concepts listed in Table 2. These are Java tools that are able to open and compile any java source files. The remaining tools allow users to implement certain of the programming concepts.

Table 2. Evaluation of programming assistance tools using selection criteria: Programming Concepts.

✓ = tool meets the criteria **● = tool can be adapted to meet the criteria**	**RoboMind**	**BlueJ**	**Greenfoot**	**Scratch**	**RobotProg**	**B#**	**Jeliot**	**Ville**	**PlanAni**	**Alice3D**	**JGrasp**
Programming Language (s=specific, n=non-specific)	n	s	s	n	n	s	s	n	n	n	s
Concepts											
Variables	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Input (getting information from the user)	●	✓	✓	✓	✓	✓	✓		✓	✓	✓
Output(displaying information to the user)	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
If-statements	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Case (Delphi)/Switch(Java) statements		✓	✓				✓				✓
For-loops	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Repeat loops	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
While loops	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
String handling		✓	✓				✓	✓			✓
Procedures	✓	✓	✓	✓	✓		✓	✓		✓	✓
Functions	●	✓	✓				✓				✓
One-Dimensional Arrays		✓	✓	✓			✓	✓	✓		✓
Two-Dimensional Arrays		✓	✓				✓				✓
File handling		✓	✓				✓				✓
Accessing a database		✓	✓				✓				✓
SQL statements		✓	✓				✓				✓
Correct use of parameters		✓	✓				✓	✓			✓
Objects & classes		✓	✓	✓			✓			✓	✓
Problem solving	✓	✓	✓	✓	✓		✓			✓	✓
Debugging	✓	✓	✓				✓	✓			✓

B#, Jeliot, PlanAni, Ville and RobotProg can assist users with code comprehension and code execution through the use of visualisation and animation (Table 3). All of the tools, except Alice and Scratch, can assist users to improve their knowledge of programming language syntax. The statements used by the original RoboMind are similar to Java but the editor can be adapted to compile statements that users are more accustomed to using in a particular programming language. All of the tools have been developed to support user understanding of programming principles and concepts. The tools also all promote self-study and self-exploration by users.

Scratch and Alice help users to create a solution correctly. In Scratch and Alice the statements used indicate to users where conditions or variables must be inserted or if other statements must be included within a loop or control structure. RobotProg allows users to create a solution using a flowchart diagram. Users are able to visualise and compare the execution of the solution using the flowchart with the actions of the robot.

All of the tools except PlanAni allow users to code or create a solution within the tools, thus catering for the kinesthetic learning preference. In PlanAni, users can only run built-in examples to understand how the code is executed. None of the tools cater for the aural learning preference. Jeliot, Ville and PlanAni include functionality to ask users questions regarding the code or provide explanations when the code is executed. This may assist users that prefer the read/write learning preference. All of the tools address the visual learning preference by using visualisation when building the code solution or during code execution.

All of the tools can be used to assist users to apply their understanding of simple exercises to more complex exercises. None of the tools explicitly scaffold the learning. In all the tools, the example exercises provided should include simple as well as complex examples of different programming concepts to assist user understanding.

The error handling item is *greyed* out for Scratch and Alice (Table 3). These tools do not require error handling or error messages to be displayed to users. The use of the drag-and-drop interface ensures that users can only use the correct statements and syntax. Error handling and compiler messages are also not applicable for PlanAni as built-in examples are used which cannot be edited by the user.

Table 3. Evaluation of programming assistance tools using selection criteria: Programming skills and knowledge.

✓ = tool meets the criteria ● = tool can be adapted to meet the criteria	RoboMind	BlueJ	Greenfoot	Scratch	RobotProg	B#	Jeliot	Ville	PlanAni	Alice3D	JGrasp
Programming skills & knowledge:											
Code comprehension					✓	✓	✓	✓	✓		
Promotes problem solving using strategies									✓		
Code execution	✓				✓	✓	✓	✓	✓		
Syntax knowledge	●	✓	✓			✓	✓	✓	✓		✓
Knowledge of programming principles & concepts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Teaching/Learning approach:											
Constructivist (promote self-study)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Feedback to guide solution creation				✓		✓				✓	
Learning Preferences:											
Visual	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Aural											
Read/Write							✓	✓	✓		
Kinesthetic (user actually codes)	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
Other:											
Simple & Complex examples (scaffolding)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Error handling	✓	✓	✓	▨		✓	✓	✓	▨	▨	✓
Simple error messages/suggestions to correct errors	✓			▨		✓			▨	▨	
Visualisation/Animation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Can Adapt?	✓					✓		✓	✓		

Only B# and RoboMind use simple error messages that try to inform users of syntax errors in the code using language and terms that are easier for novice programmers to understand. BlueJ, Greenfoot, jGrasp and Jeliot – use the standard Java compiler. The error messages are the same messages that expert programmers would receive in professional programming environments such as Netbeans or Eclipse.All of the tools that have been evaluated are freely available for use and can be downloaded, without charge, from their respective websites. The source-code of RoboMind and B# are available for modification and adaption. Ville can be set up to convert code examples into other programming languages that are not included with the initial installation. In PlanAni, although the examples are built-in, it is possible to extend these examples using a special file format. A programmer with understanding of programming concepts and principles and who is able to work with these different tools will be able to adapt these tools to cater for more programming concepts and/or different programming languages.

6. CONCLUSION

Teaching novice programmers to program requires an understanding of the difficulties of learning to program. This research study has discussed the difficulties novice programmers face when learning to program and highlighted reasons for some of these difficulties. Figure 1 and Section 2 explain the knowledge and skills required to program successfully.

Novice programmers can be assisted by the programming environments or tool in which they learn to program, especially if the programming tool is specifically designed to assist novice programmers. Before selecting a tool, however, it is important to identify what knowledge and skills a novice programmer is trying to develop. Table 2 and Table 3 (Section 5) indicate that programming assistance tools do not meet all the criteria that have been identified to assist novice programmers. It is recommended that skills and knowledge that are most important for the novice to develop (for example, problem solving, understanding of code execution, or syntax knowledge) should first be identified in order to guide the tool selection.

One should also be aware of how difficult it may be to use a tool if no explicit instruction or assistance will be provided before providing or recommending a programming assistance tool to novice programmers. A brief document describing the main interface of the tool together with sources where additional help can be found is recommended. Including simple and complex example exercises with the tool may also assist users to understand how different programming concepts can be implemented.

The selection criteria (Table 1) presented in this paper has been used in a research study to select tools for IT learners in South African secondary schools. The selection criteria will be evaluated and changed, where necessary, based on the findings of the research.

7. ACKNOWLEDGMENTS

We would like to thank the schools, teachers and learners who participated in this research study for their valuable contributions.

8. REFERENCES

[1] Al-Imamy, S. Alizadeh, J. and Nour, M.A. 2006. On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process. *Journal of Information Technology Education*, Vol. 5, 271-283.

[2] Baldwin, K. and Kuljis, J. 2000. Visualisation Techniques for Learning and Teaching Programming. In *Journal of Computing and Information Technology - CIT 8*, Vol. 4, 285-291.

[3] Bednarik, R. and Tukiainen, M. 2006. An eye-tracking methodology for characterizing program comprehension processes. In *Proceedings of the 2006 symposium on Eye tracking research & applications (ETRA '06)*. ACM, New York, NY, USA, 125-132.

[4] Bednarik, R., Moreno, A. and Myller, N. 2005. Jeliot 3, an Extensible Tool for Program Visualisation. In *Proceedings of the Koli Calling 2005: 5th Annual Finnish / Baltic Sea Conference on Computer Science Education*.

[5] Ben-Ari, M., Levy, R. and Uronen, P.A. 2000. An Extended Experiment with Jeliot 2000. In *Proc. Of the Program Visualisation Workshop,* Porvoo, Finland.

[6] Byckling, P. and Sajaniemi, J. 2006. Roles of Variables and Programming Skills Improvement. *SIGCSE Bulletin*, Vol. 38(1), March 2006, 413-417.

[7] Cooper, S., Dann, W. and Paush, R. 2003. Teaching Objects-first in Introductory Computer Science. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE '03)*. ACM, New York, NY, USA, 191-195.

[8] De Raadt, M. 2008. *Teaching Programming strategies explicitly to Novice Programmers*. Doctoral dissertation. University of Southern Queensland.

[9] Department of Education. 2008. *National Curriculum Statement. Grades 10-12 (General)*. Learning Programme Guidelines. Information Technology.

[10] Fleming, N. and Baume, D. 2006. Learning Styles Again: VARKing up the right tree! Educational Developments, SEDA Ltd., Issue 7.4, 4-7, November 2006.

[11] Garner, S. 2007. A program design tool to help novices learn programming. In *ICT: Providing choices for learners and learning. Proceedings Ascilite Singapore 2007*, 321-324.

[12] Gayo-Avello, D. and Fernández-Cuervo, H. 2003. Online Self-Assessment as a Learning Method. In *Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies*, 2003. Published 9-11 July 2003, 254-255, ISBN: 0-7695-1967-9.

[13] Greyling, J.H., Cilliers, C.B. and Calitz, A.P. 2006. B#: The Development and Assessment of an Iconic Programming Tool for Novice Programmers. In *7th International Conference on Information Technology Based Higher Education and Training (ITHET'06)*, 367-375.

[14] Haataja, A., Suhonen, J., Sutinen, E. and Torvinen, S. 2001. High School Students Learning Computer Science Over the Web. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*. Wake Forest University. Vol. 3(2), October 2001.

[15] Henriksen, P. and Kölling, M. 2004. Greenfoot: Combining object visualisation with interaction. In *Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA)*, pp. 73-82, Vancouver, BC, CANADA, November 2004.

[16] Hu, C. 2008. Just Say 'A Class Defines a Data Type'. In Communications of the ACM, Vol. 51(3), 19-21.

[17] jGRASP 2009. Overview of jGRASP and the Tutorials. DOI= http://www.jgrasp.org/tutorials187/00_Overview.pdf.

[18] Kölling, M. 1999. The Problem of Teaching Object-Oriented Programming, Part 1: Languages. *Journal of Object-Oriented Programming*, Vol. 11(8), 8-15.

[19] Kölling, M. and Rosenberg, J. 2001. Guidelines for Teaching Object Orientation with Java. In *Proceedings of the 6th conference on Information Technology in Computer Science Education (ITiCSE 2001)*, Canterbury, 2001.

[20] Kölling, M. and Rosenberg, J. 2001. BlueJ - The Hitch-Hikers Guide to Object Orientation.

[21] Koorsse, M., Calitz, A.P. and Cilliers, C.B. (2010). Programming in SA Secondary Schools: The Inside Story. SACLA, 2010.

[22] Koorsse, M. Cilliers, C.B. and Calitz, A.P. (2010). Motivation and Learning Preferences of Information Technology Students in South African Secondary Schools. SAICSIT, 2010.

[23] Lahtinen, E., Ala-Mutka, K., and Järvinen, H. 2005. A study of the difficulties of novice programmers. In *Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Caparica, Portugal, June 27 - 29, 2005). ITiCSE '05. ACM, New York, NY, 14-18.

[24] Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., and Thomas, L. 2004. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. In *Working Group Reports from ITiCSE on innovation and Technology in Computer Science Education*, Leeds, United Kingdom, June 28-30, 2004. ITiCSE-WGR '04, 119-150.

[25] Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy, June 26 - 28, 2006). ITICSE '06. ACM, New York, NY, 118-122.

[26] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. 2007. A survey of literature on the teaching of introductory programming. In *Working Group Reports on ITiCSE on innovation and Technology in Computer Science Education* (Dundee,

Scotland, December 01 - 01, 2007). J. Carter and J. Amillo, Eds. ITiCSE-WGR '07, 204-223.

[27] Pendergast, M.O. 2005. Teaching Introductory Programming to IS Students: Java Problems and Pitfalls. In *Journal of Information Technology Education*, Vol. 5, 491-515.

[28] Rajala, T., Laakso, M., Kaila, E. and Salakoski, T. 2007. VILLE – A Language-Independent Program Visualisation Tool. *Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007)*, Finland.

[29] Resnick, M., Malone, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y. 2009. Scratch: Programming for All. In *Communications of the ACM*, Vol. 52(11), 60-67.

[30] Robins, A., Rountree, J. and Rountree, N. 2003. Learning and Teaching Programming: A Review and Discussion. In *Computer Science Education*, Vol. 13(2), 137–172.

[31] Rongas, T., Kaarna, A. and Kalviainen, H. 2004. Advanced Learning Technologies. In *Proceedings of IEEE International Conference on Advanced Learning Technologies, ICALT'04*, 678–680.

[32] Shuhidan, S., Hamilton, M. and D'Souza, D. 2009. A Taxonomic Study of Novice Programming Summative Assessment. In *Eleventh Australasian Computing Education Conference (ACE2009)*, (Wellington, New Zealand, January 2009). Conferences in Research and Practice in Information Technology (CRPIT), Vol. 95. M. Hamilton & T. Clear, Eds.

[33] Utting, I., Cooper, S., Kölling, M., Maloney, J. and Resnick, M. 2010. Alice, Greenfoot, and Scratch - A Discussion. *ACM Transactions on Computing Education*, Vol. 10(4), Article 17, Pub. Date: November 2010.

[34] Vickers, P. 2009. How to Think Like a Programmer. Cengage Learning EMEA. ISBN: 978-1-84480-903-5.