The role of Visualisation in the study of Computer Programming

Sarah Alhammad sarah.alhammad@plymouth.ac.uk
Shirley Atkinson shirley.atkinson@plymouth.ac.uk
Liz Stuart L.stuart@plymouth.ac.uk

School of Computing, Electronics and MathematicsPlymouth University
&ROOHJHRI&RPSXWHUDQG,QIRUPDWLRQ6FLHQFHV3ULQFHVV1RXUDKELQWDEGXOUDKPDQ8C

Abstract

This paper presents a study of how visualization tools and methods are used to support the study of computer programming. The purpose of this study was to establish whether visualisation could be exploited more fully or more effectively to support this learning. Furthermore, if that were the case, then this study would aim to define the requirements of novice programmers. Greater understanding of the requirements of novice programmers is expected to steer this research towards finding better solutions to the challenges faced by students of programming.

The paper presents a study of three distinct visualisation tools Jeliot(Moreno, Myller, Sutinen, & Ben-Ari, 2004), Online Python Tutor (Guo, 2013) and Visual Logic (Gudmundsen & Olivieri, 2011). The nature of how visualisation is exploited in each of these tools is evaluated. This evaluation is based on how the tool visualises the execution of (i) loops, (ii) object-oriented programming and (iii) the parameter –passing by value/reference. These three problems are a sample of problems that students find complex when writing programs (Eckerdal et al., 2006; Boustedt et al., 2007; Sanders et al., 2008).

The results presented in this paper show the characteristics in the visualisation tools and the students' preferences in using them. The students evaluated the following eight characteristics: tool availability, error explanation, expression evaluation, the interface, programming languages the tool supports, animation, class hierarchy and save the execution history.

1. Introduction

Technology is central to the way we live our lives these days and programming is the "engine" behind this extensive use of technology. Many students are keen to study programming. However, one of the challenges when teaching programming is helping students to understand traditional static program code maps designed to serve as a dynamic representation of program memory (Ala-Mutka, 2004). Teachers often try to illustrate the effect of code on memory simply using paper and whiteboards. However, this can be time-consuming, and it often lacks interaction with and engagement of the student.

Visualisation tools are used in many academic institutions to support learning in many diverse areas. Programming is no exception. Indeed, it is common for teachers and students to use visual tools to trace the execution of a program as well as showing the outcome from each statement of code as it is executed. The use of visualisation tools is an alternative to the use of paper and whiteboards to demonstrate the execution of the program. However, visualisation tools deliver a more effective understanding of the execution as the students can interactively "see" exactly what changes are happening in the program memory as each line of code executes.

As the research domain of visualisation expands, more visualisation methods are emerging which present the teacher with an alternative to manual program tracing. These visualisation tools are aimed at supporting novice programmers as they learn to program (Baldwin & Kuljis, 2000,2001; Kasurinen, Purmonen, & Nikula, 2008; Salcedo & Idrobo, 2011). This research focuses on tools which visualise the program memory and enable the students to step-through the code, watching the changes in the memory as each line executes. Note that it is important to establish the effectiveness of existing visualisation tools to fully appreciate the user requirements.

This work is organized as follows: First, section 2 views studies that have been conducted to describe the challenges of learning programming. In section 3, we discuss the educational visualisation method for learning programming. Section 4 is concerned with the threshold concepts in programming. The sixth section discusses the experiment and its results. Finally ,the last two sections are about limitations of the study and the conclusion.

2. The challenges of Learning Programming

Numerous studies have investigated the difficulties in teaching and learning computer programming. For instance, Milne and Rowe (2002) conducted an experiment that demonstrated the inability of a student to absorb what is happening in the memory of the program. Another study (Husain, Tarannum, & Patil, 2013) was conducted to enhance students' problem-solving skills and improve the learning and teaching of programming. That study showed that even better students (one with higher grade point average -GPAs) struggled to apply the programming concepts they have learned in lectures. The two most challenging factors for those students were (i) in the design and (ii) in the implementation phases when they started writing programs.

There is high demand for innovations that support the teaching of programming and deal with these many inherent challenges faced by both teachers and students.

3. Visualisation

Research studies have demonstrated the advantages of using visualisation when learning to program. Naps et al. (2003) define the visualisation in education as engagement taxonomy which means demonstrate the lecture using visualisation as viewing engagement . Kelleher and Pausch (2005) describe how the visualisation environment contributes in lowering the barriers to programming such as Mechanical and Sociological barriers. Kasurinen et al. (2008) introduced the approach of using visualisation in teaching programming for the purpose of increasing motivation and decreasing students' failing or dropout rates.

Many visualisation tools have emerged with the aim of supporting novice programmers as they learn to program. Examples include BlueJ (Hagan & Markham, 2000; Kölling, Quig, Patterson, & Rosenberg, 2003), DrJava (Allen, Cartwright, & Stoler, 2002), ProfessorJ (Gray & Flatt, 2003), Jeliot3 (Moreno, Myller, Sutinen, & Ben-Ari, 2004), VIP (Virtanen, Lahtinen, & Jarvinen, 2005) , Web Tasks (Rößling, 2010), Alice (Salcedo & Idrobo, 2011) and Online Python Tutor (Guo, 2013).

Support tools often adopt visualisation to help students overcome the challenges of learning programming. Approaches which visualise code and/or use diagrams to represent the program memory have a vital contribution to the process of teaching and learning how to program.

3.1 Visualisation tools used in this study

The three sample visualisation tools used were Jeliot, Online Python Tutor and Visual Logic. These tools all use memory referencing to show the effect of execution of each line of the program memory. However, each tool has its own features. The selection of each tool was done specifically in order to ease the comparison of student feedback. However, the tools differ in many aspects such as the way in which they trace through the code, how they represent the output, how they create error explanations, the programming languages they support and whether they are available online or offline.

Jeliot is a visualisation tool for object-oriented programming. It provides dynamic visualisation for tracing the program execution (see figure 1). Jeliot builds a model of the program during execution. Therefore, students can understand the program construction. The Jeliot interface consists of four areas: the 'method frame' area, the 'expression evaluation' area, the 'constant' area and the 'instance' area. Each area is used to show the dynamic change of its components (method, expression, constant, instance). The error explanation in Jeliot highlights the line of the error and explains the reason for the error in the 'error viewer'.Jeliot also has frames where inputs are requested, and outputs are printed (Moreno et al., 2004).

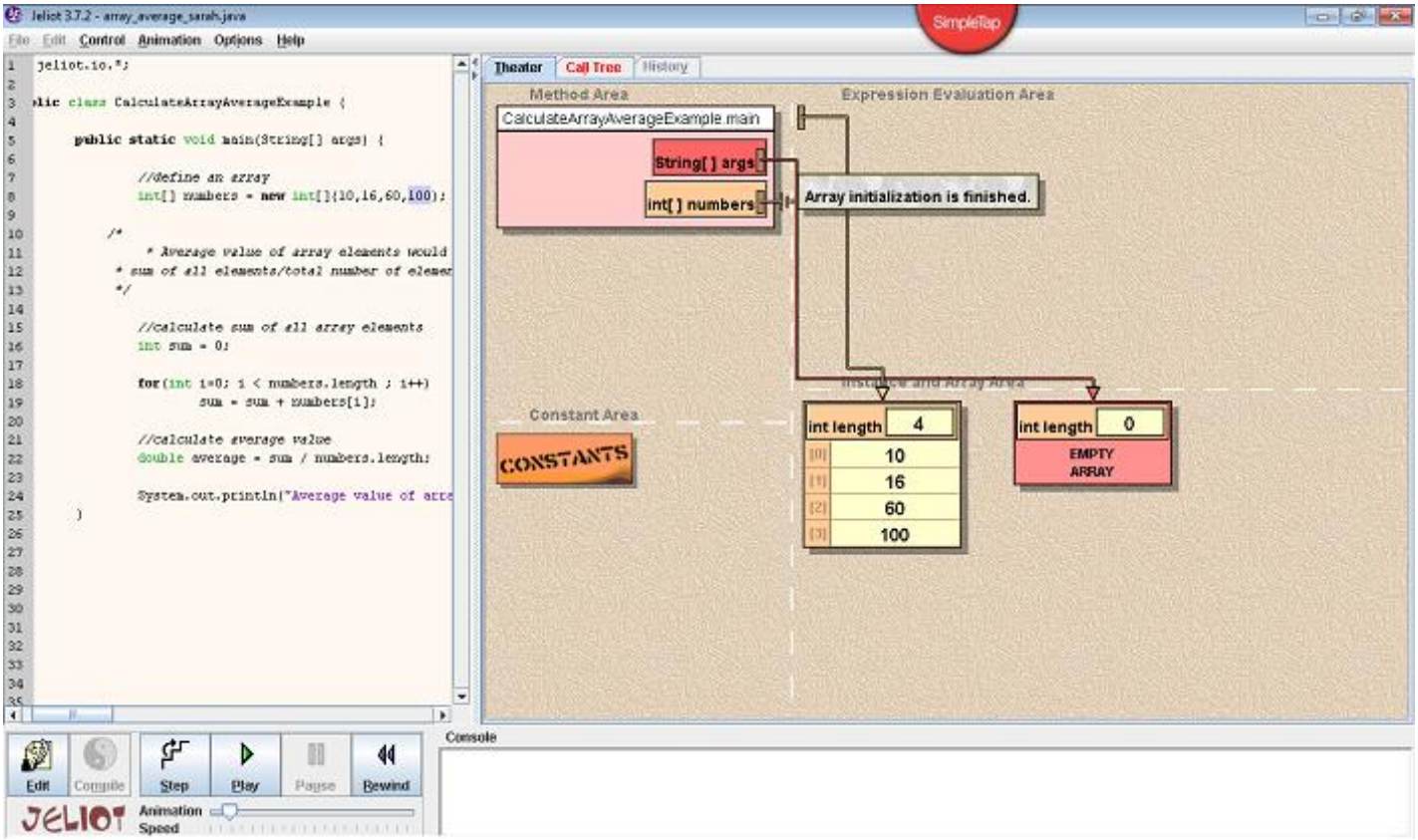

Figure 1-Jeliot

Online Python Tutor is a web-based programming tool which uses visualisation extensively. It is open-source software where the user embeds their code into the web page on the left. Subsequently, the code can be traced through using navigation buttons (see figure 2). The visualization of the code is shown on the right. This visualisation enables the user to watch the dynamic execution of the program. As the program executes it depicts changes to frames and objects. Additionally, it has a program output area. The tool provides explanations of errors, with indicators pointing to the line on which the error occurred (Guo, 2013).

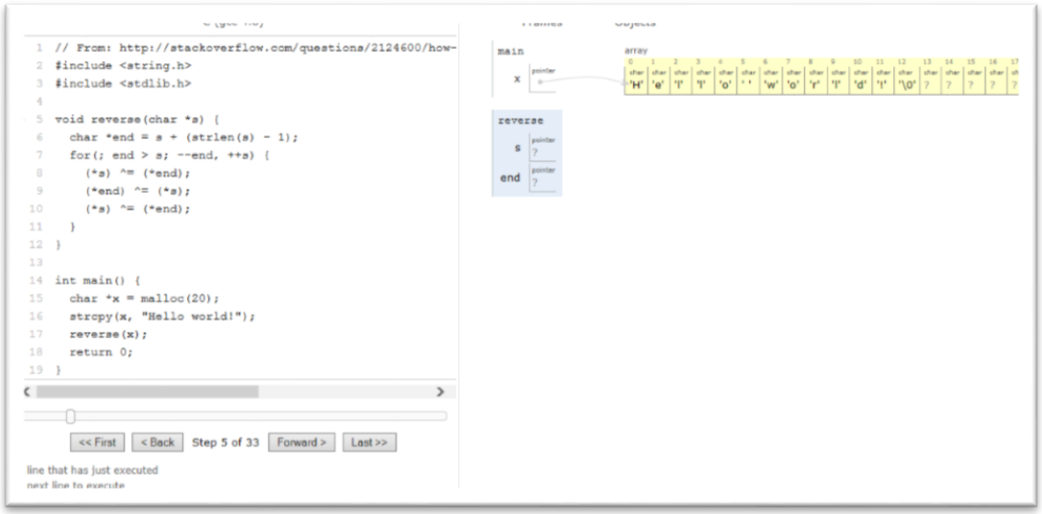

Figure 2- Online Python Tutor

Visual Logic uses the concept of iconic programming (icons and flowcharts) to visualise the program. Visual Logic has no code to be written(see figure 3). Instead, the user creates the flowchart which represents the code. Subsequently, the tool traces the flow of it. The tool demonstrates the outcomes of 'executing' each icon in the flowchart in popup windows. Visual Logic does not support object-oriented programming (Gudmundsen & Olivieri, 2011).However, the researchers have chosen the tool to introduce the use of flowchart in the tracing and observe whether this method will admire the students.

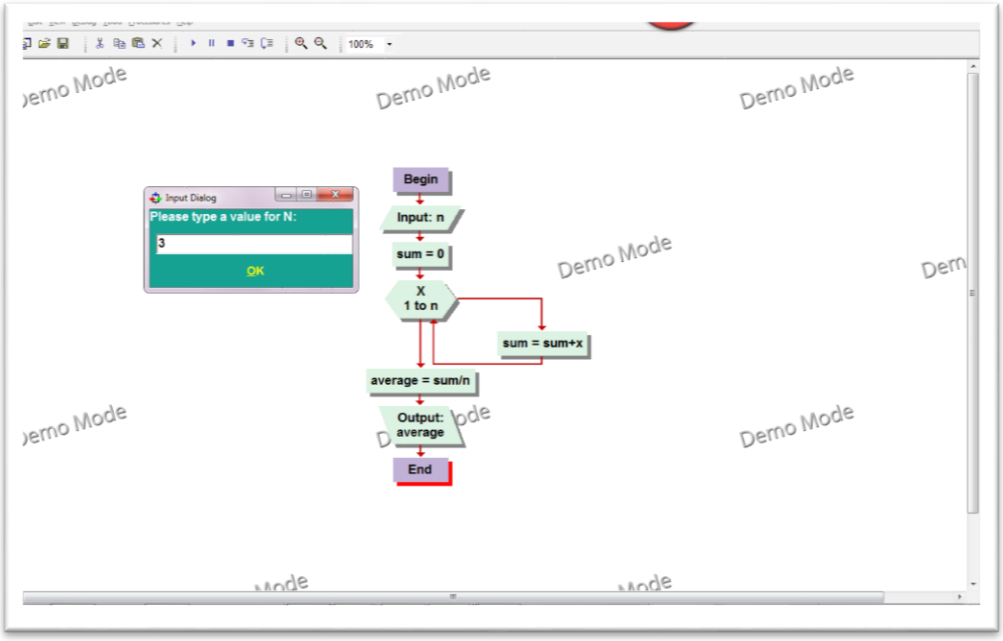

Figure 3- Visual Logic

4. Threshold concepts

Meyer and Land (2003) developed the idea of the threshold concepts in 2003. Threshold concepts are theories that link subjects together and are essential for mastering how to think and practice in the discipline. Meyer and Land (2003) defined the characteristics of threshold concepts that can be used to evaluate any scientific concepts, whether they are threshold concepts or not.

The characteristics of threshold concepts according to Meyer and Land (2003) are the following:

- Transformative: bringing about an alternate way view of things to students in the discipline.
- Integrative: linking the concepts together and exposing the interrelatedness.
- Irreversible: shifting a student's perspective and unlikely to be forgotten unless spending efforts.
- Troublesome: being hard for students to learn.
- Limited Boundary: helping students stay within the boundaries of the field and the concepts that belong to it.

The experiment, as reported in this paper, focuses on three threshold concepts used in programming. These are the use of loops, object-oriented programming and the parameters. Many research studies have investigated these concepts and determined that, among students and teachers, they are the most popular threshold concepts in computing: Eckerdal et al. (2006) found the most popular threshold concepts, among teachers, were the program's abstraction, pointers, objects, classes, instances, and recursion. In contrast, Boustedt et al. (2007) argued after the study that they had evidence that two concepts in particular—object-oriented programming and pointers—satisfy the requirements for threshold concepts. Note that Boustedt et al. also agreed on five troublesome areas of programming. These are: memory model, objects, control statements(loops), parameters, and sequential thinking.

Moreover, the findings of Sanders et al. (2008) and Bühlmann (2011) highlighted the difficulty which students have with separating the concepts of 'class', 'object' and 'instance' and with understanding the inheritance concept.

5. The study

5.1 Research methodology

This research aims to evaluate current visualisation tools used to support novice users learning to program. In particular, it will focus on identifying the strengths and weaknesses of these tools. This research was conducted on computer science students to gather information about their experience of using visualisation tools, to understand their needs better and to gather their opinions about these tools.

This study involved the use of semi-structured interviews with 20 participants, over 22 years of age. The participants were either undergraduates(second year and above) or students who had recently graduated. The interviews have been conducted in Saudi Arabia and translated to English after recording them. During interviews, it was verified that each participant had studied programming concepts in depth including the core aspects of the use of a loop, object-oriented programming and the parameters. The aim of the interview was to understand the learners' experience and to establish their background in using visualisation tools to support the process of learning to program.

Three tools were presented to the students as a sample of visualisation support tools which are available. These tools were Jeliot, Online Python Tutor and Visual Logic (as discussed in section 3.1). The students discussed the tools, their usability, and determined the strengths and weaknesses of each tool. The tools were compared on the basis of their usefulness in solving programming problems. The programming problems were selected based on the threshold concepts, including object-oriented programming, loops and parameters. Opinions were also obtained on their assessment of the best and worst tools.

5.2 Tools selection

The majority of the students were expected to have little or no experience using visualisation tools. The researcher used three samples of visualisation tools: Jeliot, Online Python Tutor, and Visual Logic. The tools all share the common method of using memory referencing to show the effect of execution on each line in the code. However, each tool has its own specific features. The tool selection was done carefully to make the comparison of students' points of view clearer. The tools differ in many aspects, such as in the way they trace code, how they represent the output, how they create error explanations, programming languages they support and whether they are available online or offline. Moreover, the tool selection was done based on the students' experiences. The tools are designed for novice programmers, which were suitable then for our participants, and require no training or special skills.

5.3 Example problems

The problem areas considered in this study are the three programming concepts described earlier as threshold concepts. Many studies (Boustedt et al., 2007; Bühlmann, 2011; Eckerdal et al., 2006; Sanders et al., 2008) have shown that loops, object-oriented programming and the parameter are threshold concepts that students find difficult to understand. Therefore, each of these core threshold concepts is embedded into the three problems investigated in this study.

5.3.1 Example problem 1 (the loop)

For this problem, Program1 was coded. Initially, this program populated an integer array data structure of five elements with predefined integer values. The overall aim of Program1 was to evaluate the average value of the integers stored in the array. To achieve this, Program1 established the number of elements in the array. Then, the program used a 'FOR loop' to calculate the sum total of all the values. Subsequently, the average was calculated.

5.3.2 Example problem 2 (the object-oriented program)

For this problem, Program2 was coded. The program used the inheritance from the superclass (polygon) to calculate the area of the subclass (square). The main program defines an instance of the

class square and passes the length of the side to the method 'get_area()', which is inherited from the superclass (polygon). The area was calculated by the inherited method "get_area()" and the result will be inherited to the subclass (square). Finally, the result printed in the main program as an element of the instance from (square) class.

5.3.3 Example problem 3 (the parameter –passing by value/reference)

For this problem, Program3 was coded. Problem 3 is used to highlight the difference between passing parameters by value and passing by reference. The main program defined variable (parameter) as a global variable and send its value to the procedure (change_value) as pass by value and not by reference. The value of (parameter) has been changed locally in the procedure but the change has no effect on the (parameter) in the main program. The variable was printed in the main program to show whether or not the value of the variable had been changed.

5.4 Data Analysis Strategy

The researcher's intention was to collect feedback about the existing visualisation frameworks to determine their benefits to develop a new framework. The qualitative data support the research to focus on the basic features and aspects that novice students need on the educational tools.

This study analysed the coding process through initial and final coding using Cycle Pattern Coding Method .This method of coding was chosen to find the keywords and group the features. Text analysis was used to conduct the first cycle of coding to determine phrases that were common amongst interviewees. Phrases appeared as word clouds, which were analysed and encoded with suitable category labels.Next, a Second cycle of the Cycle Pattern Coding Method was used to recognize similarly coded data and further summarize it into subcategories or consolidate it. Finally, findings were narrated as they related to the implications of the study. For more details of the method see (Strauss & Corbin, 2008) and (Saldaña, J. 2015).

6. Results

During the semi-structured interviews, a substantial amount of data was recorded from the participants including what they liked and disliked about each tool. During the analysis of this data, it was realized that the majority of this feedback was concentrated around eight core characteristics of support tools. All of these characteristics were features of the presented tools. Therefore, the researcher adopted this list of eight characteristics as a means of categorising the analysis of the data. The list of characteristics was

1. Controlling the execution of the code
2. Availability of the tool (online or offline)
3. Error explanation
4. Interface/usability of the tool
5. Programming languages supported
6. Expression evaluation
7. Representation of Class hierarchy
8. Maintaining an event history.

The following explains how the participants described the tool's characteristics. The number of respondents (*n*) is indicated in the headings.

6.1 Controlling the execution of the code (n=18)

The analysis stage of this study has revealed that the manner in which the participant was able to control the execution of code was significant. When this study was designed, careful attention was given to the selection of which support tools to include in the study. The outcome of this selection was that the tools chosen represented a variety of mechanisms for the control of execution. Furthermore, the participants realised that there were different kinds of execution control and they subsequently used them to compare the tools.

During the interviews, the participants used a variety of phrases to describe the control of execution. These phrases included: the animation, the control buttons, the execution speed, tracing and visualisation.

Participant 5 complained about the inability to have full control of the execution, such as the ability to go back and forth with each statement. Participant 5 said: '*The execution control in not enough, I prefer if I can control the execution more, for example, going back and forth for each statement.*' Conversely, Participant 19 appreciated the animation used to visualise the execution. They stated: '*I like the animation used to visualise the execution*'.

Regarding the speed of the execution and the ability to control it, Participant 7 suggested a slower speed: '*I want to slow down the execution speed*' . Furthermore, Participant 11 suggested an overall more flexible execution process which gave more control to the user. Participant 11 stated: '*I like the control buttons that make the control of execution more flexible.*'

The interviews demonstrated that the students clearly understood the importance of how the visualisation was being controlled. The majority of the students, 90% of them (18 out of 20), reported that they preferred to have precise controls for the line by line execution of the code and it's visualisation.

However, these preferences did vary. Of these 18 participants, 7 participants appreciated the code animation which had few controls. Conversely, 5 participants preferred the user to have complete control using the control buttons. The remaining 6 participants (30% of the participants) suggested that a mixture of both methods would be the optimal solution. This was based on the fact that the type of control required was about the length of code involved and the user's familiarity with the code.

6.2 Availability of the tool (n=16)

Secondly , the data analysis revealed that establishing whether the tool was available online and/or offline is a relevant aspect to participants. It was clear that this characteristic would influence their choice of tool. However, for some participants the online availability of the tool was a genuine concern, as shown in the feedback:

- '*I do not like the online tool, because of any internet issues or problems'* [Participant 1].
- *'I don't like the online tools to avoid the connection problems'* [Participant 7].

Conversely, for others, online availability was assessed as a positive aspect of the tool: 'I like that the tool is online tool' [Participant 17].

Thus, answers given by participants did vary on this characteristic. In total, 16 of the 20 participants highlighted this characteristic as important. Of these 16, 3 expressed a preference for the tool to be downloaded to their devices. They explained that their preference was due to concerns about their Internet speed and connection. Conversely, 9 participants preferred the online mode, explaining that they would have access to the tool at anytime and anywhere. The remaining 4 participants preferred the option of a mixed mode (combining both online and offline) where the user can choose the mode required.

6.3 Error explanation (n=13)

Thirdly, the data analysis revealed that the explanation of errors and support for debugging are important aspects to participants. The participants reported that they required (i) support for finding an error, (ii) a meaningful explanation of the error and (iii) a suggestion of a suitable means to correct the error. They highlighted that ambiguity in the explanation of the error may impede their understanding of the cause of the error in code as shown in the following feedback:

- '*The error explanation was not good'* [Participant 10].
- '*The correction suggestion for the error is not helpful enough'* [Participant 17].

Subsequently, this may have a negative effect on their learning. When the participants were asked about the reason for their opinions, they said that they could not understand the cause of the error and that the suggestions were ambiguous. Therefore, clarity in error explanation and removal are a high

priority. In total, 13 of the 20 participants highlighted this characteristic as important. All of these 13 participants expressed a preference for a support tool which explains the error in detail and provides a variety of ways to correct the error.

6.4 Interface/Usability of the tool (n=13)

The data analysis revealed that the style and appearance of the interface were another important aspect to participants. The participants commented on the font type, size and colour used in the text: '*The code font is small*' [Participant 9].

Some of the feedback gathered from the participants was about how the windows and how the tool presented the final output: '*I like the appearance of execution in Jeliot which is next to the code directly'* [Participant 1].

The participants also gave varied comments about the use of an indicator icon to point out the statement currently being executed in the Online Python Tutor. For instance, Participant 2 stated they liked '*The use of green and red pointers (arrows) to indicate the statement execution whether it is under execution or will be the next statement to be executed'* . This opinion was reinforced by Participant 6 who agreed *'I like the use of red and green arrows as an indication of statement execution.'*

Out of the total number of 20 participants, 13 participants (65%) reported on the interface and its usability from different perspectives.

Note that they used different terminologies to describe their opinion of the interface of each tool including font, colour, window, ease of use. In conclusion, it is clear that the colour, the font, and the windows' appearance played a significant role in attracting the students to use the tool.

6.5 Programming languages supported (n=9)

Next, the data analysis revealed that both the number and range of programming languages supported by a tool is important to participants. Again, the participant's feedback was mixed. Some participants preferred to use multi-programming language tools while other participants expressed a preference for a bespoke tool for each programming language.

- *'I don't like that Jeliot supports only one programming language'* [Participant 10]
- *'I like the Online Python Tutor tool because it supports more than one language'* [Participant 18]

Out of the total number of 20 participants, 9 participants (45%) recommended a tool which could be used for multiple programming languages. The remaining 11 (55%) reported that the choice of programming language was not a huge issue to consider and that supporting only one language is enough when it comes to using the tool.

6.6 Expression evaluation (n=5)

Next, the data analysis revealed that details on how and when an expression is evaluated are important aspects to participants. The participants reported that they had not recognised when expressions were being evaluated, although there was a variety of feedback on this characteristic.

For example, in Program1 where the condition of the loop is evaluated, the participants wanted to 'see' the working out of evaluating the condition of the loop at every iteration.

- *'The design needs to be developed, similar to the manual tracing, and the absence of statement expression'* [Participant 4].
- '*One of the strengths of the tool is the existence of expression evaluation'* [Participant 18].

Participants reported that this evaluation of expressions was primarily used to understand 'what's going on.' Clearly, this is a vital aspect for increasing the learners' comprehension.

In total, 5 of the 20 participants highlighted this characteristic as important. The participants reported that they relied on the expression evaluation for evaluating the condition of control statement. All of the participants who reported the use of the expression evaluation agreed that they liked it because it is similar to what they had done on manual tracing. Therefore, there was a requirement to present the user with the automated version of what they would otherwise do manually.

6.7 Representation of Class Hierarchy (n=2)

The analysis stage of this study has revealed that the manner in which the tool represented the class hierarchy was critical to users.

Example problem 2 presented the participants with code from an object-oriented program. That program is an example of the concept of inheritance as the program inherits variables and methods from a predefined superclass. The participants assessed how well each tool represented the classes and their hierarchy. They also reported how much the tool aided their comprehension of the classes in the code.

For example, Participant 1 made the following statement about Jeliot stating '*in case of representing the classes and inheritance, it is not clear because the classes cascaded and not represented as hierarchy.'* Participant 3 was agreed '*There was no weakness in the tool, but then went onto to highlight one exception 'its way to represent the class hierarchy and the variables and methods which are inherited or private all of these was not satisfied.'*

6.8 Maintaining an event history (n=2)

The final characteristic of support tools which was identified as important by study participants was the tools' ability to record the events (a history) of everything that took place during execution. The participants used the phrase 'save history' to describe their requirement for the system to automatically generate a list of events which took place throughout the whole process of execution.

Participant 2 complained about the tools' being unable to save the history. Participant 3 also supported this by indicating that they would also like a tool that keeps results for each execution and does not omit events. Based on this feedback, it is clear that users place a high value on having this history of events to trace through after execution.

6.9 Tool comparison

The final part of this study focused on a comparison of the three tools. This comparison was based on how each tool visualised the execution of the three problems (mentioned in section 5.3) and then gathering the participants' opinions about which tools they preferred.

The students excluded the Visual Logic tool from the comparison since it received a lot of negative feedback on its usefulness for support students learning to program. It was a clear outlier that they were not interested in pursuing. The reason for this negative feedback was that Visual Logic does not include any program code and that it relies too heavily on tracing through the flowchart.

Regarding example problem 1, solving the FOR-LOOP problem, all of the 20 participants preferred the Jeliot tool. The reason for this unanimous decision was that Jeliot has 'expression evaluation'. This 'expression evaluation' feature shows the evaluation of the condition of the loop for each iteration. Furthermore, it shows the loop counter and the loop body. Participant 3 stated: '*I choose Jeliot, in case I am novice programmer because it has the feature which is expression evaluation*.' Furthermore, Participant 4 stated: '*Jeliot was clearer than the other tool, I like its way on how it shows the expression evaluation to clarify the loop counter, loop condition, and the body of the loop.'* These opinions were reinforced by other participants:

- *'Jeliot was clearer because it represents the dynamic change on the loop counter and condition'* [Participant 7]
- *'Jeliot is better because the loop was clearer in how the counter change each time and how the loop condition checked every time'*[Participant 15]

Regarding example problem 2, based on object-oriented programming, 7 participants expressed a preference to use Jeliot. They attributed this choice to the fact that cascading the classes is more effective than representing them on a hierarchy. This is particularly the case when there are numerous classes where the screen space will be insufficient to demonstrate the class hierarchy. Participant 1 said: '*Jeliot is better in case of long code and plenty of classes because the class cascaded which make space add classes.'* Participant 4 said: '*I prefer Jeliot because the way on how representing the classes were clear specifically if the number of classes was large.'*

Conversely, the majority of the participants(11 out of 20) preferred the Online Python Tutor tool as it represents the classes sequentially. Therefore, the participants reported that they 'could see' the variables and methods of each class. Participant 10 said: '*Online Python Tutor because the classes were presented sequentially so it is clearer than representing the classes on Jeliot'*. Participant 18 reported that: '*Online Python tutor is better because I like how it is drawn boxes sequentially for objects and use arrows to represent the references.'*

Only two of the participants were impartial. Those two participants stated that both class representations were clear and helped them to understand the classes inheritance equally.

Finally, regarding example problem 3, the researcher asked the participants to compare the tools when calling procedures and passing parameters. The vast majority of the participants, 18 out of 20, agreed that the Online Python Tutor tool was best. The reason they gave was that the process of parameter passing was significantly clearer than in the Jeliot tool.

There were different opinions based on the clarity provided by the Online Python Tutor tool. For instance, Participant 2 stated that '*Online Python Tutor is clearer on how the transition done from the main to the procedure and how it affect the value of parameters.'*

Participant 11 said: '*Online Python Tutor was clearer when calling procedures and giving the returning value, it always writes what is the return value from any procedure even if its "void"*.'

However, the remaining two participants said they understood the problem equally when using both Online Python Tutor and Jeliot.

7. The limitations of the study

The study has explored three threshold concepts while there are other threshold concepts which could be explored such as pointers and recursion. Building a visualisation tool to support learning programming should probably account for more than the three concepts .

Moreover, some of the characteristics that have been found in the study mentioned by very few participants for example expression evaluation, representation of class hierarchy and maintaining an event history. Whilst these characteristics do seem fairly important for a learning system. The claim in the study suggesting that they are critical to users should be weakened however we need more supporting data for this.

8. Conclusion and future work

The aim of this study was to assess how well current visualisation systems support students as they learn to program. Overall, the coverage of threshold topics is variable between systems. This study identified a set of 8 core characteristics used to assess support tools. In essence, the following list is essential requirements for tool support in this area. These requirements include a tool which:

- is highly flexible and enables the user to use it for 1 or more languages (as required)
- is available online and offline
- is intuitive based on good design of the interface which delivers a usable product
- has a clear evaluation of expressions in the code (as in a manual trace)
- has a clear, meaningful and helpful explanation of errors with support for error correction
- has an intuitive control of the execution of the program code,
- provides an clear intuitive representation of the class hierarchy which can be compacted when a number of classes is large

- maintains an event history for the user to trace through and search.

Future work in this research aims to develop a software tool based on the findings of this study. The aim is to learn from existing systems and to develop a more complete, flexible tool for supporting students who are learning to program.

Finally, there are some factors that could serve as implications for the study. Firstly, the data collection period was time limited, and due to the fact that the research method involved semi-structured interviews (as opposed to a case study), the interviewers had a limited period to gather students' opinions. To perform a more in-depth evaluation, the students would need a longer time to practice using the tools and becoming familiar with them. Some of the participants in the study wanted to use the tools for longer periods in order to provide more accurate feedback. Secondly, the researcher aims to extend the study to cover a variety of international sites. Different kinds of cultures and different academic institutions, outside of the Saudi Kingdom, may enrich the study with valuable results and findings.

9. References

Ala-Mutka, K. M. (2004). Problems in learning and teaching programming-a literature study for developing visualizations in the Codewitz-Minerva project. Codewitz Needs Analysis, 1–13.

Allen, E., Cartwright, R., & Stoler, B. (2002). DrJava: A lightweight pedagogic environment for Java. ACM SIGCSE Bulletin, 34, 137–141. doi:10.1145/563340.563395

Baldwin, L. P., & Kuljis, J. (2000). Visualisation techniques for learning and teaching programming. ITI 2000. Proceedings of the 22nd International Conference on Information Technology Interfaces (Cat. No.00EX411). doi:10.2498/cit.2000.04.03

Boustedt, J., Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., Sanders, K., & Zander, C. (2007). Threshold concepts in computer science. ACM SIGCSE Bulletin, 39(1), 504. doi:10.1145/1227504.1227482

Bühlmann, M. (2011). T He Q Uality of D Emocracy ; C Rises and S Uccess S Tories, 49(1), 123–128.

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., Sanders, K., & Zander, C. (2006). Putting threshold concepts into context in computer science education. ACM SIGCSE Bulletin, 38(3), 103. doi:10.1145/1140123.1140154

Gomes, A., & Mendes, A. J. (2007). An environment to improve programming education. Proceedings of the 2007 International Conference on Computer Systems and Technologies. ACM, 1. doi:10.1145/1330598.1330691

Gray, K., & Flatt, M. (2003). ProfessorJ: a gradual introduction to Java through language levels. Companion of the 18th Annual ACM SIGPLAN …, 170–177. doi:10.1145/949344.949394

Gudmundsen, D., & Olivieri, L. (2011). Using Visual Logic © : Three Different Approaches, 23–29.

Guo, P. J. (2013). Online python tutor: Embeddable web-based program visualization for cs education. SIGCSE 2013 - Proceedings of the 44th ACM Technical Symposium on Computer Science Education, 579–584. doi:10.1145/2445196.2445368

Hagan, D., & Markham, S. (2000). Teaching Java with the BlueJ environment. Proceedings of Australasian Society for Computers in Learning in Tertiary Education Conference ASCILITE 2000.

Husain, M., Tarannum, N., & Patil, N. (2013). Teaching programming course elective: A new teaching and learning experience. IEEE International Conference in MOOC, Innovation and Technology in Education (MITE), 275–279. doi:10.1109/MITE.2013.6756349

Kasurinen, J., Purmonen, M., & Nikula, U. (2008). A Study of Visualization in Introductory Programming. Ppig '08, (Winslow 1996), 181–194.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys, 37, 83–137. doi:10.1145/1089733.1089734

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ system and its pedagogy, 13(4), 1–12. doi:10.1076/csed.13.4.249.17496

Meyer, J., & Land, R. (2003). Threshold concepts and Troblesome knowledge:linkages to ways of thinking and practising within the disciplines.

Milne, I., & Rowe, G. (2002). Difficulties in Learning and Teaching Programming — Views of Students and Tutors. Education and Information Technologies, 7, 55–66. doi:10.1023/A:1015362608943

Moreno, A., Myller, N., Sutinen, E., & Ben-Ari, M. (2004). Visualizing programs with Jeliot 3. Proceedings of the Working Conference on Advanced Visual Interfaces - AVI '04, 373. doi:10.1145/989863.989928

Naps, T. L., Rodger, S., Velázquez-Iturbide, J. Á., Rößling, G., Almstrum, V., Dann, W., … McNally, M. (2003). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, *35*(2), 131. doi:10.1145/782941.782998

Rößling, G. (2010). A family of tools for supporting the learning of programming. Algorithms, 3, 168–182. doi:10.3390/a3020168

Salcedo, S. L., & Idrobo, A. M. O. (2011). New tools and methodologies for programming languages learning using the scribbler robot and Alice. Proceedings - Frontiers in Education Conference, FIE, 1–6. doi:10.1109/FIE.2011.6142923

Saldaña, J. (2015). The coding manual for qualitative researchers. Sage.

Sanders, K., Boustedt, J., Eckerdal, A., McCartney, R., Moström, J. E., Thomas, L., & Zander, C. (2008). Student understanding of object-oriented programming as expressed in concept maps. ACM SIGCSE Bulletin, 40(1), 332. doi:10.1145/1352322.1352251

Strauss, A., & Corbin, J. (2008). Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Basics of Qualitative Research Grounded Theory Procedures and Techniques (Vol. 3). doi:10.4135/9781452230153

Virtanen, A. T., Lahtinen, E., & Jarvinen, H.-M. (2005). VIP, a Visual Interpreter for Learning Introductory Programming with C++. Koli Calling '05.