Design and Use of Static Scaffolding Techniques to Support Java Programming on a Mobile Phone

Chao Mbogo
Department of Computer Science
Kenya Methodist University
Nairobi, Kenya
chaombogho@gmail.com

Edwin Blake
Department of Computer Science
University of Cape Town
Cape Town, South Africa
edwin@cs.uct.ac.za

Hussein Suleman
Department of Computer Science
University of Cape Town
Cape Town, South Africa
hussein@cs.uct.ac.za

ABSTRACT

Most learners in resource-constrained environments own mobile phones that they could use to learn programming while outside the classroom. However, limitations of mobile phones, such as small screens and small keypads, impede their use as typical programming environments. This study proposed that programming environments on mobile phones could include scaffolding techniques specifically designed for mobile phones, and designed based on learners' needs. Scaffolding should be designed with some essential techniques that are mandatory for learners to use. Hence, one type of scaffolding technique that was designed to support programming on the mobile phone is static scaffolding that does not fade. Experiments were conducted with 64 learners of programming from three universities in Kenya and South Africa in order to investigate how they used the designed static scaffolding techniques to construct Java programs on a mobile phone. The results show that programming on mobile phones can be supported by providing scaffolding techniques that never fade, in order to address the limitations of mobile phones and to meet learners' needs.

Keywords

Mobile phone; Java; Programming; Static Scaffolding.

1. INTRODUCTION

The learning difficulties encountered in computer programming [23], especially by novice learners, indicate that some programming skills are beyond the novice learners' efforts. Scaffolding refers to support provided so that learners can engage in activities that would otherwise be beyond their unassisted efforts [24]. In order to contribute towards tackling learning difficulties in programming, novice learners can be supported to learn programming while they are outside the classroom. This makes any such support additional to the learner's classroom learning, and not a replacement.

Support to learners outside the classroom can be provided using PC-based applications. However, in many developing countries, people are much more likely to use computers at school or at work than to own them at home. For example, a survey conducted in Ghana and Kenya to investigate the ownership of information and communication technologies at home showed that only 10% of respondents in Ghana and 5% in Kenya have a computer at home [2]. The limited access to PCs outside the classroom aggravates the learning difficulties in the subject.

The ubiquity of mobile devices provides an opportunity to use them as a resource to support learning of programming beyond the classroom. Mobile devices include laptops, tablets and mobile phones. Of these, mobile phones are the most widely used mobile devices among learners in developing countries [11]. Therefore, the mobile phone was selected as the resource that can be used for construction of programs outside the classroom. However, limitations of mobile phones, such as a small screen size and a small keypad, impede their use as typical programming environments. To deal with these limitations, and for handheld devices to become effective learning tools, the unique design challenges inherent in such a system must be understood [14]. In addition to addressing limitations of mobile phones, the challenges faced by learners of programming should be considered. This is because addressing these challenges maximizes the potential of meeting learners' needs. Consequently, this study proposed that programming environments on mobile phones could include scaffolding techniques that are specifically designed for mobile phones, and designed based on learners' needs.

One design recommendation is that scaffolding should be designed with some essential character that provides mandatory scaffolding to support learners [18]. For example, essential scaffolding was implemented in the design of a PC-based environment that provided a process wheel, which is a process map that visually described the space of possible science inquiry activities that learners could select from [17]. The design of such scaffolding that does not fade was encouraged because such scaffolds help to focus learners' attention and also ensure that a consistent, basic level of support is provided for every learner [20]. In this study, such scaffolds that do not fade are termed as static scaffolding.

Static scaffolding was designed as one of three types of scaffolding techniques to support Java programming on a mobile phone. The other two are: (i) automatic scaffolding that is automatically provided but fades with time or can be cancelled by the user; (ii) user-enabled scaffolding that is not automatically provided and the learner has to initiate its use. This paper focuses on static scaffolding. To implement the scaffolding techniques, an Android prototype was developed that supports the construction of Java programs on a mobile phone [15]. Android was selected as the platform of implementation because it is open source. Java was selected as the language for construction of programs because it was the common language taught across the institutions that participated in the study.

1.1 Designed Static Scaffolding Techniques

Static scaffolding techniques were designed using a theoretical scaffolding framework [16] that provides two strategies to support their design: (i) providing visual organizers in order to give access to functionality; and (ii) constraining the space of activities by using functional modes and by using ordered or unordered decomposition.

Providing visual organizers in order to give access to functionality was implemented by designing a program layout of

ITiCSE '16, July 09-13, 2016, Arequipa, Peru
 ISBN 978-1-4503-4231-5/16/07…$15.00
DOI: http://dx.doi.org/10.1145/2899415.2899456

the parts of a Java program. The order of the program layout was guided by standard Java coding guidelines [9], where a Java source file has the following ordering: beginning comments, package and import statements, and class and interface declarations. Figure 1 shows the designed main interface with parts of a Java program. This layout uses clickable buttons that provide: (i) collapsible and expandable views such as in Figure 1, where the main class button has been clicked to reveal some default code within the expanded area; and (ii) access to create individual parts of the program by clicking inside the expanded area. Such a collapsible and expandable interface was recommended for small screens [3].

Constraining the space of activities by using functional modes and decomposition was implemented by enabling construction of a program one part at a time. In the main interface (Figure 1) the learner clicks on the button that relates to the part they need to work on. Figure 1 shows only the main class as enabled and can be constructed at this stage. Until the learner correctly creates the main class the other parts of the program remain disabled. Thereafter, the learner is guided to create the header comments part then the main method part and so on. The program layout is retained even when learners progress to an advanced interface, where the order of program creation is not restricted. Thus, the program layout is a static scaffolding technique since it does not change or fade away with time.

On clicking each program part on the main interface another interface is opened with an editor that provides creation of only the selected program part. For example, Figure 2 shows creation of only the main method. The ability to work on one part of the program at a time could assist in working with the small screen. Because of the restriction of a small screen size, which remains unchanged, this scaffold is static and does not fade. Further, Figure 2 shows how working on a program one part at a time could assist in addressing the soft keypad on smartphones that takes up nearly half the screen.

For a learner to have a mental image of how the different parts of the program work together, learners should be able to inspect the task they are working on in multiple ways. In this case, while working on a program part (for example, while editing the main method in Figure 2), a learner could swipe to the full program interface and view the whole program at the state at which it was last saved (Figure 3). This ability to move between a program part and the whole promotes cognitive growth by keeping the learner connected to the program parts, while at the same time being able to appreciate existence of the whole problem [1].

To compile the program at any time, the learner presses the button at the top right corner of Figure 1 and the full program is sent to the ideone online compiler and debugging tool [8]. The results and output are sent back to the mobile interface.

To evaluate the use of the static scaffolding techniques, an empirical evaluation was conducted where 64 learners from three universities in Kenya and South Africa attempted Java programming tasks using the application. Data was collected using computer logs and questionnaires.

The contribution of this study is fourfold: (i) an illustration of static scaffolding techniques that do not fade; (ii) how the static scaffolding techniques support the construction of Java programs on a mobile phone; (iii) feedback from learners on the use of static scaffolding techniques; and (iv) implications of the study.

2. RELATED WORK

Studies on supporting the learning of programing stress the importance of learning programming by doing, which is in line with the constructivist theory. Learning programing by doing requires access to computing resources such as PCs and laptops. Indeed, several studies have offered scaffolded environments on PC platforms targeting novice learners of programming, for example, 3D environments such as Alice [4], and teacher-learner assessment environments such as Test My Code [22]. However, most learners at institutions in parts of Africa are in resource-constrained environments where they have limited access to such resources, especially while they are outside the classroom. Even within the institutions, some schools have a limited number of desktop computers that could be shared among learners. For example, even in a relatively well-resourced developing country like South Africa, it is not uncommon for a school of 1,000 learners to have only one computer room with 30 PCs [20]. In fact, poor infrastructure and facilities is one of the major challenges faced by higher education in Africa [25]. This study was motivated by the resource constraints in a developing country's context.

The ubiquity and availability of mobile phones provides an opportunity to use them to support learning of programming outside the classroom. A study conducted in Kenya showed that most of the respondents studying for university degrees or higher own mobile phones [7]. However, mobile phones pose some limitations. The key limitation of handheld technology for the delivery of learning objects is the small screen that is available [3].

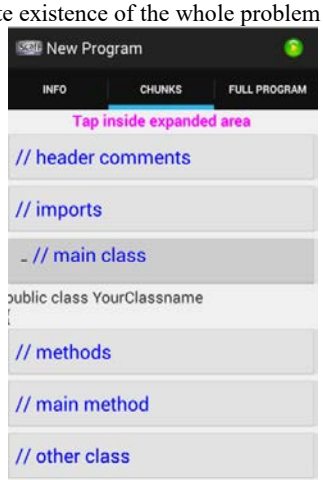

Figure 1. Main interface showing program overview with only the main class parts activated

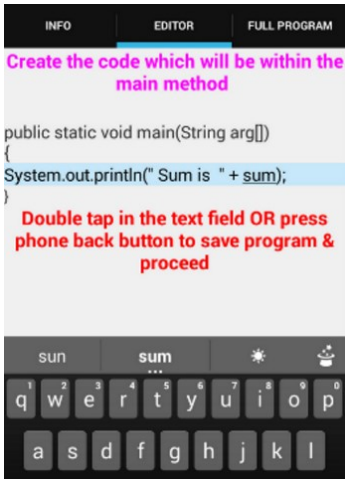

Figure 2. Editor interface showing construction of only the main method

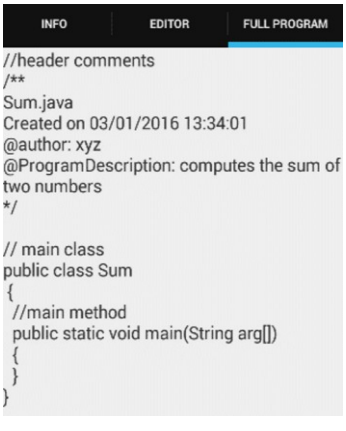

Figure 3. Full program as was last saved

One recommendation for designing scaffolds is by using activity decomposition that develops separate workspaces for each component task [13] to package contents in small program parts [5]. Such design recommendations were considered while designing the static scaffolding techniques discussed in this paper.

There are existing mobile programming environments that can be used by novice learners. Some, such as SAND IDE, can be used to create standard programs. However, mobile programming environments such as SAND IDE mostly mimic PC IDEs and do not address the limitations of mobile phones. A study by Microsoft enables development of applications using a new language - TouchDevelop - on the TouchDevelop programming environment where much of the code is created by tapping through menus [19]. TouchDevelop is a specialized language that was designed for a visual programming environment that creates mobile applications. In contrast, this study does not develop a specialized language. Further, it was not the aim of this study to support the creation of mobile applications, but to support the creation of standard programs that would typically be created in an introduction to programming class.

3. STUDY METHODOLOGY

3.1 Participants and Experiments

Table 1 shows the distribution of 64 learners of programming from one university in South Africa (University of Western Cape (UWC)) and two universities in Kenya (Jomo Kenyatta University of Agriculture and Technology (JKUAT) and Kenya Methodist University (KeMU)) who participated in two experiments. Participation in the experiments was voluntary. Experiment one was conducted with a group of learners different from the ones in Experiment two. Despite the geographical and background differences between South Africa and Kenya, all learners were taking an introductory course in programming using Java. The learners who took part in this study used desktop IDEs in their classroom learning. However, during the experiments they only used the mobile programming interface.

Each group of learners from each university took part in 2-hour experiment sessions. Each experiment session involved an introduction to the purpose of the research with learners signing consent forms, learners tackling the programming tasks, and completion of a post-experiment questionnaire. Learners who did not own Android phones were issued with such phones with the application pre-installed. The phones issued were the Samsung Galaxy Pocket S5300 phones that run Android version 2.3. The Samsung Galaxy Pocket has a display size of 2.8 inches.

3.2 Programming Tasks

The teachers were asked for a set of Java exercises relating to introductory topics that they had already taught in the course. Three sets of programming tasks were used during the two experiments: one set of questions for UWC in Experiment one; one set of questions for JKUAT in Experiment one; and one consolidated set of exercises for both KeMU and JKUAT in Experiment two. In the first Experiment, the exercises were obtained from the different teachers of the courses in their respective institutions. In the second Experiment, the teachers from both KeMU and JKUAT had taught similar topics in introduction to Java programming. Therefore, the exercises from the respective teachers were combined into one set. At the time of conducting the two experiments, all the teachers had covered the topics of Java syntax, input-output, loops, methods, and classes. The programming tasks attempted by learners in Experiment one are shown in Figure 4. The programming tasks attempted by learners in Experiment two are shown in Figure 5.

Table 1. Distribution of learners in two experiments

Experiment	Institution	Number of learners
One	UWC	14
	JKUAT	13
Two	KeMU	13
	JKUAT	24

Programming Task for UWC group in Experiment One

1. Write a program that calculates the total cost of an item that is R159.72 and incurs a VAT of 14%.
2. Write a program that uses a for-loop to calculate the sum of the numbers from 1 to 50 and displays the sum and average.
3. Write a program that uses a method name() to print out your name.
4. Write a program that uses the Scanner input to ask for the user's name and age, and prints
   "Hello " + name " your age is "+ age;
5. Write a program that uses a method input() to ask for height and width of a rectangle, and calculates and display the area using height x width.
6. Write a program that determines if a number that is input by a user is odd or even.

Programming Task for JKUAT group in Experiment One

1. Write a program that outputs 'Scaffolding at JKUAT'.
2. Write a program that computes the sum and average of the number 1-20.
3. Write a program that captures and displays the ages of two students.
4. Write a program that uses a method to capture two integers and outputs their sum.
5. Write a program that initialises default values of name and age in a constructor and outputs these in a main class.

Figure 4. Programming tasks attempted by learners in Experiment one at UWC and JKUAT

1. Write a program that initialises x to 10 and prints out its double value. Save this program as XValue.java
2. Using a for-loop print the first 10 natural numbers. Save this program as Natural.java
3. Write a program that accepts input from the user and displays this as
   "Your input is " + input. Save this program as Natural.java
4. Write a program that uses a method input() to capture and display the names of two students. Save this program as MethSt.java
5. Write a program that creates two classes. The second class contains the constructor below. Access this constructor from the main class
   Output() { System.out.println ("Constructor called"); }
6. Write a program that uses a for-loop within a method avg() to calculate the sum of the numbers 20-100 and displays the sum. Call this method from the main method.

Figure 5. Programming tasks attempted by learners in Experiment two

3.3 Data Collection

Google Analytics was used to collect logs of the learners' interaction with the application. At the end of the experiments the learners filled an online questionnaire that consisted of two parts: (i) demography; and (ii) reflections and perceptions on scaffolding techniques.

4. Evaluation

The CIAO model [10] and the micro and meso levels of the M3 evaluation framework [21] have outlined that while evaluating educational technology one should consider data about learners' interaction with the software and learners' attitudes and outcomes. Thus, in order to investigate the use of static scaffolding techniques, three criteria were considered: (i) task success; (ii) the use of the static scaffolding techniques to construct programs; and (iii) qualitative feedback from the learners.

4.1 Task Success

Each program was examined for the extent to which it was completed. A complete program is one that met all three criteria: (i) had all the required program parts completed; (ii) successfully compiled after completion of the required parts; and (iii) produced the required output. Four metrics measured task success: (i) which tasks were attempted; (ii) which tasks were not attempted; (iii) which tasks were incomplete; and (iv) which tasks were completed. Incomplete tasks are tasks that failed to meet at least one of the criteria for completeness. Attempted tasks are the combination of incomplete and completed tasks. Some tasks were not attempted.

4.2 Use of Static Scaffolding Techniques

Three metrics measured the use of static scaffolding techniques: (i) use of static scaffolding techniques in incomplete and complete programs; (ii) progression of use of static scaffolding techniques from one task to the next; and (iii) learners' characteristics while using the static scaffolding techniques.

4.3 Qualitative Feedback

Qualitative feedback was collected using self-reported data by learners reflectively indicating their perceptions on the use of static scaffolding to support construction of programs on a mobile phone.

5. Results and Discussion

This section presents results and discussion on the use of static scaffolding techniques, some characteristics displayed by learners while using the static scaffolding techniques, and representative learners' feedback. In the graphs, UWC-1 means the first experiment at UWC, KeMU-2 means the second experiment at KeMU, and so on.

5.1 Use of Static Scaffolding Techniques

Static scaffolding was provided using two techniques: (i) a program overview that also offered restricted program creation in the basic main interface; and (ii) editing of a program one part at a time while able to view the full program. Figure 6 shows a comparison of the use of static scaffolding techniques in complete and incomplete programs across the four experiment sessions in the first and second experiments. The average use per learner refers to the average number of times that each learner accesses the interfaces that provide each of the two static scaffolding techniques. Figure 6 shows that there was variation in use of the static scaffolding across the experiments. For example, in the first experiments at UWC and JKUAT, learners who completed programs edited the program parts more than the learners who did not complete programs. Whereas in the second experiment at KeMU, learners who did not complete programs edited the program parts more than the learners who completed programs. This variation in use could be because learners had to interact with the static scaffolds to construct the programs, whether or not they completed the programs successfully. In all the cases learners spent more time on average on the program overview than on editing the parts of a program. This could be because the program overview interface is the entry point to all the program parts and a learner had to go back to this interface in order to access each program part. Conversely, the editing interface involved working on just one program part a time.

Additional analysis was conducted on the use of static scaffolding across the different tasks. The results from the second experiment at JKUAT are used to illustrate this because it is the group where the most number of tasks were attempted and completed (Table 2 shows the number of learners who attempted and completed each task at JKUAT in the second experiment). Figure 7 shows the progression of use of static scaffolding from the first program to the sixth program. Learners used the static scaffolding nearly two times less in the second program than in the first; meaning that learners spent less time both on the main interface and working on the program parts in the second program than in the first. The reduced use of the static scaffolding in the second program could be due to learners having familiarized themselves with the interface. Figure 7 also indicates that the static scaffolding was mostly used in the first program than in subsequent programs for both incomplete and complete programs. Some of the programs that were completed in the fourth task were constructed at the advanced interface. This explains the increased use of static scaffolding since learners encountered this interface for the first time. Further, all the tasks that were completed in the sixth program were completed within the advanced interface. These tasks required the construction of a method in addition to the main class, header and main method. This explains the increased use of static scaffolds at the sixth program. These results indicate that, indeed, learners were able to attempt and complete programming tasks using the static scaffolding techniques, which had to be used for all programs.

5.2 Learners' characteristics while using static scaffolding techniques

Further comparison of the use of static scaffolding with the use of automatic scaffolding (such as instructions and prompts for examples) and user-initiated scaffolding (such as hints on program parts) revealed that learners found static scaffolding alone sufficient to construct programs. Three examples will be used to illustrate this.

While creating a program part for the first time, some learners repeatedly went back to the editor on the same program part, before proceeding to the next one. For example, 7 learners in the first experiment at UWC exhibited this characteristic. In contrast, there were learners who initially worked on each program part just once or made at most two attempts before proceeding to the next program part. The common characteristic among such learners is that they mostly used only the static scaffolding techniques with partial use of some of the provided automatic scaffolding and very little use of the user-enabled scaffolding. This is evidence that the static scaffolding techniques are sufficient to support construction of programs on a mobile phone, even when the learners do not use scaffolding that they can choose (user-enabled) or that which fades.

Table 2. Number of learners who attempted and completed tasks at JKUAT in the second Experiment

	Attempted	Completed
Task 1	24	18
Task 2	19	17
Task 3	20	12
Task 4	12	7
Task 5	6	3
Task 6	5	3

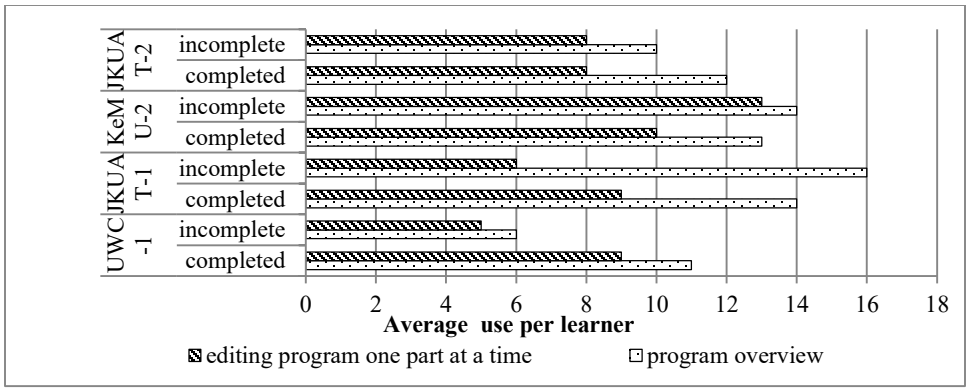

Figure 6. Comparison of use of static scaffolding techniques between incomplete and complete programs at UWC, KeMU and JKUAT in Experiments one and two

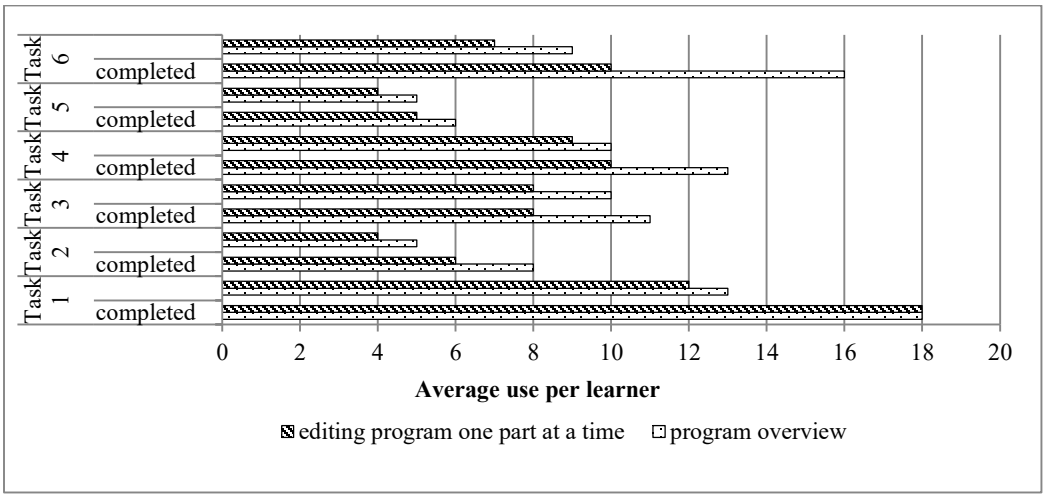

Figure 7. Progression of use of static scaffolding techniques in incomplete and complete programs at JKUAT Experiment two

Another illustration is when a learner was working on a program, where a suggestion to view a related example was provided (an option that a learner could accept or reject). These were automatic scaffolds. It was observed that several learners opted not to view these examples. For example, of the 24 learners in the second experiment at JKUAT, 18 rejected the use of one scaffolding technique or another, with 11 learners rejecting a suggestion to view an example. This suggests that learners may not have required extra support such as viewing of examples, but found it sufficient to use only the static scaffolding to create programs.

Further evidence that static scaffolding supports construction of programs on a mobile phone was observed by how learners edited programs after they encountered run-time errors. After learners encountered run-time errors, they were able to go directly to the part of the program that contained the erroneous code by easily accessing it through the program layout at the main interface.

5.3 Learners' feedback on the use of static scaffolding techniques

Learners found the two static scaffolding techniques useful as evidenced by the representative verbatim feedback: 'The application divides the program or code into sections then one can then track and write the code properly by following the sections.' 'The sections are well laid out.' 'The separate segments of program are useful.' 'How the codes are divided into chunks making the application easier to use.' 'The chunks made it easier to construct the program.'

The learners' representative positive feedback further indicates their usefulness in supporting programming on a mobile phone.

6. Conclusion

This study has presented two static scaffolding techniques: (i) a program layout at the main interface; and (ii) editing of a program one part at a time while able to view the full program. The results show that the program layout and constructing a program one part at a time enabled effective support and guidance towards correct creation of programs. Further, learners' verbatim feedback indicate that they found these static scaffolding techniques useful.

Desktop IDEs provide complex environments where a large amount of information is exposed to the learner at the same time, because this is possible on such large screens. This also means that it is possible to provide support to the learner all in one place without the learner having to leave the screen. However, providing all the functionality in one place does not work well on small screens. One technique that was used in this study to address the small screen is the static scaffolding technique of completing a program one part at a time. This way, the learner is able to focus on only the small part and correctly create it, hence learn it, before learning the next small part. This study has given an indication that the benefits of a static scaffolding technique such as completing a program one part at a time may not have been achieved if such as scaffolding technique was optional.

One of the main criticisms of the constructivist approach is that learners are expected to construct new knowledge with minimal guidance [12]. This criticism was discussed by Guzdial [6], where he posed the question: how then should programming be taught considering that the emphasis has been to learn programming by constructing programs? This study provides one possible answer.

The static scaffolding techniques designed in this study provide strong guides by ensuring that there is always support available that address the limitations of mobile phones and learners' needs. Thus, one possible answer to Guzdial's question is: learning programming on such small devices can be supported by providing some static scaffolding techniques that are always present. In addition, in resource-constrained environments where it may not be possible to easily acquire new desktops for learners that they could use outside the classroom, the solution could be to use the devices that the learners already have and design applications that consider both the limitations of the available devices and learners' needs. This study has shown that this is possible.

Future work from this study will include several aspects: (i) use of the scaffolding techniques to attempt more complex and larger programs than the ones presented in this paper, such as programs with multiple methods, controlled loops, or inheritance; (ii) the limitations of the interface encountered when tackling more complex and larger programs and if and how these might influence the design of additional scaffolding techniques; (iii) a comparative study with a desktop programming environment (with and without scaffolding); (iv) a study involving pre-test and post-test analysis in order to test if learners gained programming skills; and (v) a longitudinal study where learners use the static scaffolding over an extended period of time.

7. Acknowledgements

We thank the learners, teachers and institutions that participated in this study. We thank the Kenya Education Network (KENET) and Kenya Methodist University for travel grants to attend the conference.

REFERENCES

[1] Ackermann, E.K. 1996. Perspective-Taking and object Construction. *Constuctionism in Practice: Designing, Thinking, and Learning in a Digital World*. 25–37.

[2] Bowen, H. and Goldstein, P. 2010. *Radio , Mobile Phones Stand Out in Africa ' s Media Communication Landscape*.

[3] Churchill, D. and Hedberg, J. 2008. Learning object design considerations for small-screen handheld devices. *Computers & Education*. 50, 3 (Apr. 2008), 881–893.

[4] Dann, W.P. et al. 2011. Learning to Program with Alice. (Mar. 2011).

[5] Elias, T. 2011. Universal instructional design principles for mobile learning. *The International Review of Research in Open and Distributed Learning*.

[6] Guzdial, M. 2015. What's the best way to teach computer science to beginners? *Communications of the ACM*. 58, 2 (Jan. 2015), 12–13.

[7] Hannah, B. 2010. *Information at the Grassroots: Analyzing the media use and communication habits of Kenyans to support effective development*.

[8] Ideone ™ API: 2010. .

[9] Java Code Conventions: 1997. *http://www.oracle.com/technetwork/java/codeconventions-150003.pdf*. Accessed: 2015-01-19.

[10] Jones et al. 1999. Contexts for evaluating educational software. *Interacting with Computers*. 11, 5 (May 1999), 499–516.

[11] Kafyulilo, A. 2012. Access, use and perceptions of teachers and students towards mobile phones as a tool for teaching and learning in Tanzania. *Education and Information Technologies*. 19, 1 (Jul. 2012), 115–127.

[12] Kirschner, P.A. et al. 2006. Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist*. 41, 2 (Jun. 2006), 75–86.

[13] Luchini, K. et al. 2004. Design guidelines for learner-centered handheld tools. *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04* (New York, New York, USA, Apr. 2004), 135–142.

[14] Luchini, K. et al. 2002. Supporting learning in context: extending learner-centered design to the development of handheld educational software. *Proceedings. IEEE International Workshop on Wireless and Mobile Technologies in Education* (2002), 107–111.

[15] Mbogo, C. et al. 2013. A mobile scaffolding application to support novice learners of computer programming. *Proceedings of the Sixth International Conference on Information and Communications Technologies and Development Notes - ICTD '13 - volume 2* (Cape Town, Dec. 2013), 84–87.

[16] Mbogo, C. et al. 2014. Initial Evaluation of a Mobile Scaffolding Application that seeks to Support Novice Learners of Programming. *To appear in Proceeding of Mobile Learning 2014 Conference* (Madrid, Spain, 2014).

[17] Quintana, C. et al. 2002. A Case Study to Distill Structural Scaffolding Guidelines for Scaffolded Software Environments. *Proceedings of the SIGCHI conference on Human factors in computing systems Changing our world, changing ourselves - CHI '02* (New York, New York, USA, Apr. 2002), 81.

[18] Quintana, C. et al. 2002. Scaffolding Design Guidelines for Learner-Centered Software Environments. (Mar. 2002).

[19] Tillmann, N. et al. 2011. TouchDevelop. *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software - ONWARD '11* (New York, New York, USA, Oct. 2011), 49.

[20] Traxler, J. and Vosloo, S. 2014. Introduction: The prospects for mobile learning. *PROSPECTS*. 44, 1 (Apr. 2014), 13–28.

[21] Vavoula, G. and Sharples, M. 2009. Meeting the Challenges in Evaluating Mobile Learning: A 3-level Evaluation Framework. *International Journal of Mobile and Blended Learning*. 1, 2 (2009), 54–75.

[22] Vihavainen, A. et al. 2013. Scaffolding students' learning using test my code. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education - ITiCSE '13* (Canterbury, England, Jul. 2013), 117.

[23] Watson, C. and Li, F.W.B. 2014. Failure rates in introductory programming revisited. *Proceedings of the 2014 conference on Innovation & technology in computer science education - ITiCSE '14* (New York, New York, USA, Jun. 2014), 39–44.

[24] Wood, D. et al. 1976. The Role of Tutoring in Problem Solving. *Journal of Child Psychology and Psychiatry*. 17, 2 (Apr. 1976), 89–100.

[25] Yizengaw, T. 2008. *Challenges of Higher Education in Africa and Lessons of Experience for the Africa-US Higher Education Collaboration Initiative*.