Supporting the Construction of Programs on a Mobile Device: A Scaffolding Framework

Chao MBOGO[1], Edwin BLAKE[2] , Hussein SULEMAN[3]
[1,2,3]*Department of Computer Science, University of Cape Town, South Africa*
Email: chao.mbogo@uct.ac.za[1], edwin@cs.uct.ac.za[2], hussein@cs.uct.ac.za[3]

Abstract: Computer programming is a difficult subject for most novice learners. Providing support that complements classroom learning could contribute to tackling the difficulties. Due to the ubiquity of mobile devices, such support can be provided by scaffolding the construction of programs on a mobile device. In order to design such a mobile intervention, learners' needs and limitations of mobile devices need to be placed at the center of the design process. This paper combines learners' needs and limitations of mobile devices to identify scaffolding strategies. Identification of scaffolding strategies is based on a scaffolding framework. Using specific examples, this paper will then show how the scaffolding strategies have been implemented on a mobile device.

Keywords: Computer Programming, Scaffolding, Mobile, Framework

1. Introduction

Computer programming is a difficult subject for most novice learners. Research indicates this to be a universal problem (Apiola et al., 2011) (Maleko et al., 2012). This paper forms part of research that aims to contribute to tackling challenges among novice learners of programming, especially in resource-constrained environments.

Scaffolding refers to support provided so that the learner can engage in activities that would otherwise be beyond their abilities (Jackson et al., 1998). Providing such support, in addition to the learners' classroom learning, could

contribute to tackling learning challenges. Supporting learners outside the classroom recognizes that learning can take place at any place, in any situation. Therefore, the aim is to make the most of the resources available to support learning, by making their provision more flexible, open and responsive to the needs of individual learners (Bentley, 2012).

The ubiquity of mobile phones provides an opportunity to use them as a resource to support flexible learning beyond the classroom. Moreover, mobile phones could be used to support learning in cases where a learner does not own a personal computer while away from school, or is in a situation where using a personal computer would be inconvenient. In addition, recent work by Microsoft on TouchDevelop (Tillmann et al., 2012) indicates that a programing environment that runs on a mobile device has the potential to dramatically reduce the technical learning overhead.

In order to design such a mobile intervention, learners' needs need to be understood, which helps in informing the design of an intervention that seeks to support them. In addition to addressing learners' needs, designing such a mobile intervention requires that the limitations of mobile devices be addressed. This is because, in order for handheld devices to become effective learning tools, the unique design challenges inherent in such a system must be understood (Luchini et al., 2002). Significant research has been carried out to propose guidelines for designing on mobile devices for learning (for example (Luchini et al., 2004) (Churchill & Hedberg, 2008) (Elias, 2011)). This paper will refer to some of these studies in order to address design issues for supporting construction of programs on a mobile device.

Having identified learners' needs and limitations of mobile devices, the next task is to propose scaffolding strategies that could address them. To achieve this, this study utilizes a 5-step scaffolding framework that culminates in implementing the scaffolding strategies on a mobile device.

The rest of the paper is organized as follows: using 3 examples, section 2 reports on challenges and covers both learner-cited challenges and mobile limitations; section 3 consolidates the learners' needs and mobile limitations within a scaffolding framework in order to identify scaffolding strategies, and illustrates how these scaffolding strategies have been implemented on a mobile device; and Section 4 concludes the paper.

2. Challenges

2.1 Learner-cited challenges

In order to understand the needs of programming learners, an online survey was conducted among 160 learners of programming from three African universities: University of Cape Town (UCT) (61 learners); University of Western Cape (UWC) (37 learners); and Kenya Methodist University (KeMU)

(62 learners). The three universities were chosen because of their convenience in terms of having established contacts. The survey was conducted by sending an electronic questionnaire to the learners. The targeted learners were all from computer related courses because programming is part of their curriculum.

76% of the total respondents indicated one challenge or the other that they have faced or do face while learning programming. The learners were also asked if they had used a mobile phone to construct programs, with 99% of the learners indicating that they have not. For the sake of providing detailed illustration, the 3 learner-cited challenges below are randomly selected from the ones cited, and will be used as running examples for the rest of the paper.

i. Difficulty in combining required program parts into a working program and hence making logic or sense out of a program is challenging. This challenge is further supported by research pointing to two key problems preventing success in programming among novice learners (Guzdial et al., 1998): *decomposition problem,* where learners have difficulty choosing which of the available program components are needed for a solution; and *composition problem,* in which even when learners identify program components, they have difficulty assembling the modules into a proposed solution.
ii. Unclear error messages while debugging. A study that looked at common Java errors made by learners (Hristova et al., 2003) indicates that even though compilers may flag some of the error messages while programming, often the error messages are so cryptic to students that they have a hard time understanding them.
iii. Small screens of mobile devices pose a challenge in using it as a resource to learn programming. This limitation is described in the next subsection.

2.2 Mobile phone limitation

There are certainly several factors that have to be taken into consideration when it comes to mobile devices since they present usability problems (Kukulska-Hulme, 2005) (Kukulska-Hulme, 2007). However, to define the scope of which mobile limitations to consider, this paper will look at screen size (as pointed out by the learners) and the small keypad. Considering these limitations is crucial because, in writing a program, a learner needs to see (on a screen display) what they are constructing (through typing).

Small screen and keypad size

The key limitation of handheld technology for the delivery of learning objects is the small screen that is available for effective display (Churchill & Hedberg, 2008). Research indicates that the following strategies could address the small screen sizes of mobile phones while designing for learning:

i. Using activity decomposition to structure handheld tools (Luchini et al., 2004) and package contents in small chunks (Elias, 2011).

ii. Design interface elements to serve a dual role by providing both functionality and scaffolding (Luchini et al., 2004).
iii. Minimize scrolling as much as possible (Churchill & Hedberg, 2008). Scrolling can be reduced by placing navigational features near the top of the pages in a fixed place (Jones et al., 1999).
iv. Provide one step interaction, which can be achieved by immediate update upon interacting with a widget or a button. (Churchill & Hedberg, 2008).
v. Design to include movable, collapsible, overlapping and semitransparent interactive panels (Churchill & Hedberg, 2008).
vi. Use focus and content visualization technique. Users can view local information they are interested in (focus) in details on a segment of the screen, while other peripheral information (context) is shown in the surrounding area with reduced granularity of detail (Adipat & Zhang, 2005).

The small keypad of mobile phones also presents a usability challenge. While typing is needed to write a program, automating some tasks could minimize the disadvantage of having to type on a small keypad. However, care should be taken not to have an interface that is too automated such that students complete the task by rote rather than mindfully engaging and learning about the task (Luchini et al., 2004).

Mobile devices with touch screens have a soft keypad that pops up when typing, hence covering up nearly half the screen. Minimizing scrolling by use of a tabbed screen (such that users can scroll across and not downwards), could reduce the amount of information that gets covered up by the soft keypad. In addition, using activity decomposition such that smaller tasks are presented on the screen could also mean that most, if not all of the task is visible at the top half of the screen.

3. Scaffolding framework

Having identified learner challenges and mobile limitations, the next step is to integrate them within a scaffolding framework. The scaffolding framework comprises a 5-step framework that follows the following phases:

i. Step 1: Identify learner challenges.
ii. Step 2: Categorize each learner challenge into one of three types of cognitive challenges (Quintana et al., 2009): *Sense making*, which involves the basic operations of interpreting data; *Process management*, which involves strategic decisions in controlling an inquiry process; and *Articulation and reflection*, which is the process of constructing, evaluating and articulating what has been learnt.
iii. Step 3: Identify what kind of scaffolding type the learner challenge may need, from three types (Jackson et al., 1998): *Supportive scaffolding*, which offers support for doing the task while the task itself remains

unchanged; *Reflective scaffolding*, which offers support for thinking about the task; and *Intrinsic scaffolding*, which offers support that changes the task itself and reduces complexity.

iv. Step 4: Identity the scaffolding guideline based on which the intended tool can modify the task to help learners overcome obstacles. Seven scaffolding guidelines exist (Quintana et al., 2009) and are redefined to fit into this study towards a mobile strategy for supporting learners of programming:
    a. Guideline 1: Use representation and language that bridge learners' understanding of programming.
    b. Guideline 2: Organize the mobile strategy around the semantics of the programming language.
    c. Guideline 3: Use representations that learners can inspect in different ways to reveal important properties about underlying data.
    d. Guideline 4: Provide structure for complex tasks and functionality.
    e. Guideline 5: Embed expert guidance about programming practices.
    f. Guideline 6: Automatically handle routine tasks.
    g. Guideline 7: Facilitate ongoing articulation and reflection during program construction.

v. Step 5: In this step, specific scaffolding strategies are chosen to be implemented on the mobile device in order to support construction of a program.

The next subsections discuss how steps 2 to 5 can be applied to the 3 learner challenges, leading to the selection of specific scaffolding strategies as possible solutions. The strategies identified are implemented on a mobile device developed for the Android platform. Android has been selected for development because it is open source. Java has been selected as the language of program creation within the application. This is because it is the common language taught across the 3 universities where the online survey was conducted. In addition, most, if not all universities offer a first-year programming course taught using Java.

3.1 Difficulty in connecting program parts into one

Step 2: Cognitive type

This learner challenge is one of *sense making* because it involves being able to make sense out of a program and its constituent parts, while it is also one of *process management* because it requires scaffolding strategies that can control the learner's inquiry process, that is construction of a program.

Step 3: Scaffolding type

A *supportive* scaffolding type can be provided to provide support while the learner is attempting to make sense of the different parts and functionality of a

program. At the same time, an *intrinsic* scaffolding type can be provided to reduce the complexity while creating the program.

Step 4: Scaffolding guideline

In order to support the learner in trying to make sense out of a program and its constituent parts, *using representation and language that bridge learners understanding*, and *using representation that learners can inspect in different ways* could be used as scaffolding guidelines. In order to reduce complexity while the learner is creating the program, *providing structure for complex tasks and functionality* could be used as a scaffolding guideline. These three scaffolding guidelines can be met by the *scaffolding strategies* described next, as possible solutions to support a learner to connect the different parts of a program into one.

Step 5: Scaffolding strategies and implementation

3.1.1 Provide visual organizers in order to give access to functionality

This strategy can be implemented by providing a layout of the parts of a Java program in order to give the learner an overview. The order of the parts in the interface is guided by standard Java coding guidelines (Sun-Microsystems, 1997), 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 interface with program parts that can support the kind of programs written in a beginner Java class.

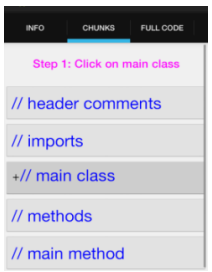

Figure 1. Main interface

Figure 2. Restricted order

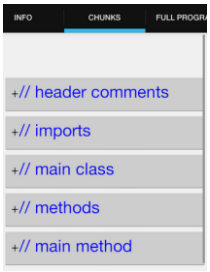

Figure 3. Unrestricted order

3.1.2 Restrict a complex task by setting useful boundaries for learners

In order to provide structure for complex tasks and functionality, this strategy can be implemented by restricting a learner to complete a program in a certain order. For example, a learner can be guided to first complete the main class because it is also used as the name of the program; then the header comment in order to guide the learner to give the description of the program they are about to write; then the main method as the entry point of the program; then they can complete methods and import sections if needed.

Figure 1 shows only the main class activated when the program is started, while Figure 2 shows the main class completed (in green) and the header comment is activated. After completion of a certain number of programs in this restricted order, a learner can be presented with an interface where all the parts are enabled and the learner is able to complete the program in any order (Figure 3). A similar technique has been used in a recent study of

scaffolding a programming course on a PC (Vihavainen et al., 2013), where fading of a restrictive scaffold is realized by using open exercises that do not enforce any specific program structure or approach.

While the learner can work with the interface in Figure 3, they are able to go back to the restricted interface if they wish to. This allows for the scaffold to fade but a learner can enable it by choice. This also provides structure to complete the task using ordered decomposition (restricted) and unordered decomposition (unrestricted) (Quintana et al., 2009).

3.1.3 Embed expert guidance to help learners use content

In order to bridge the learner's understanding with standard coding structure, or what they are learning in the classroom, some default code can be provided. Figure 4 shows implementation of default code in creating the main class declaration, which the learner can then edit (Figure 5). Figure 6 shows implementation of default code in creating a method that the learner can then edit (Figure 7). Additionally, the learner's coding history can be saved and such text can then be reused, as done in TouchDevelop (Tillmann et al., 2012).

Further, in order for the learner to be able to know which page they are working on or which one to swipe to, the different pages of the application can be labeled at the top as shown in figures 4 to 7.

Figure 4. Main class

Figure 5. Edit main class

Figure 6. Method

Figure 7. Edit method

3.2 Difficulty in debugging errors in programs

Step 2: Cognitive type

This learner challenge is one of *process management* because it requires scaffolding strategies that can contribute to the learner's inquiry process, that is, debugging of a program. It is also one of *articulation and reflection* because it contributes to thinking about and evaluating what has been constructed.

Step 3: Scaffolding type

An *intrinsic* scaffolding type can be provided as error prompts to reduce the complexity while debugging the program. This also offers a *reflective* scaffolding type that enables the learner to think about the program.

Step 4: Scaffolding guideline

In order to support process management, the intervention should *embed expert guidance about the scientific practice*, in this case being Java coding guidelines. In

order to support articulation and reflection, the intervention should provide *ongoing articulation and reflection* during completion of the program. These two *scaffolding guidelines* can be met by the *scaffolding strategies* described next.

Step 5: Scaffolding strategies and implementation

3.2.1 Embed expert guidance to clarify characteristics of Java practices

While a novice learner constructs a program, they will inevitably make mistakes that will lead to compile time or run time errors. While it is not possible to predict all the types of mistakes that learners can make, this study will attempt to address Java-syntax related errors. This is because learners indicated syntax to be a difficulty in the subject, and another study indicated Java programming syntax as among the top 5 difficulties while learning programming (Sivasakthi & Rajendran, 2011).

Figure 8 shows creation of a main class, albeit using an incorrect syntax of starting a Java class name in lower case. If the learner proceeds with this class name creation, then an error message is displayed (Figure 9). If they choose to not edit the class name, they can exit this window but no changes will be made to the class name, hence it will be considered as not having been created.

Figure 10 shows creation of a main method. Assume a learner erroneously writes the return statement here, an error prompt will be displayed indicating this error (Figure 11).

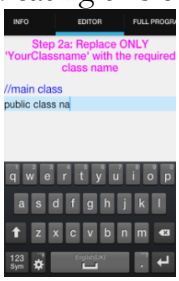

Figure 8. Main class

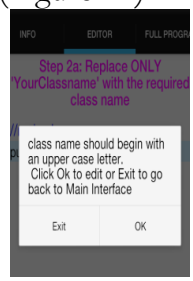

Figure 9. Prompt

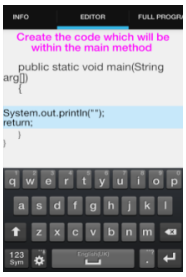

Figure 10.Main method

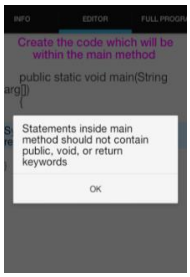

Figure 11. Prompt

3.3 Small screen size and small keypad of a mobile device

Step 2: Cognitive type

This learner challenge is one of *process management* because it requires scaffolding strategies that can support the learner's inquiry process on a mobile device, which has these limitations.

Step 3: Scaffolding type

A *supportive* scaffolding type can provide support while the learner is using the small screen size and small keypad.

Step 4: Scaffolding guideline

In order to support the learner in using the mobile device with these limitations, *providing structure for complex tasks*, and *automatically handling non-salient and routine tasks* could be used as scaffolding guidelines. These two scaffolding

guidelines can be met by the *scaffolding strategies* described next, as possible solutions to support use of mobile devices with small screen and keypad limitations.

Step 5: Scaffolding strategies and implementation

3.3.1 Constrain the space of activities by using functional modes but enable inspection of multiple views of the same object or data

A task can be scaffolded by enabling the program to be completed one part at a time. Because of the restriction of small screen size, which will remain unchanged, this scaffold is static and should not fade. Ability to work on a part of the program at a time uses activity decomposition to package the small chunks (Luchini et al., 2004) (Elias, 2011). This assists in working with the small screen. Figures 8 and 10 show how working on one program part at a time could assist in addressing the soft keypad taking up nearly half the screen, and minimize scrolling. By placing the task to be edited near the top of the screen, the soft keypad does not cover much of the task, if at all. In addition, Figures 12 and 13 also show use of navigation labels at the top of the screen as recommended (Jones et al., 1999). Navigation tabs constrain the space of activities by placing information in different segments that can be viewed by scrolling across and not downwards.

However, for a learner to have a mental image of how the different parts of the task 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 editing the main method in figure 12 to add a call to the method out()), a learner can swipe to the next tab and view the whole program (Figure 13) at the state at which it was last saved. This ability to move between a program part and the whole is one of diving-in and stepping out, and promotes cognitive growth by keeping the learner connected to the chunks, while at the same time able to appreciate existence of the whole problem (Ackermann, 1996). Ability to view the whole program by swiping to the next tab enables focal and content visualization (Adipat & Zhang, 2005).

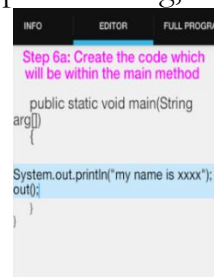

Figure 12. Editing main method

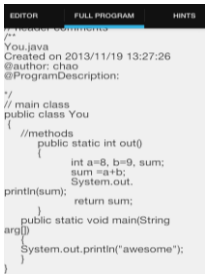

Figure 13. Full program as was last saved

3.3.2 Automate non-salient portions of tasks

Because of provision of some default code (for example, Figure 4 and 6), the learner is at least spared from typing from scratch using the small keypad.

However, the learner is still required to complete the program parts and hence they need to mindfully engage and hence learn the task, as recommended (Luchini et al., 2004). Further, the learner should be able to exit without completing a program part, but a message indicating that the task has not been changed could assist in making sure that a learner actually completes a task for it to be created in the program (Figure 14).

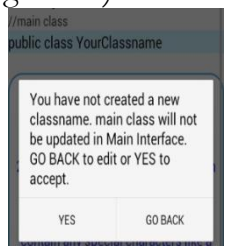

Figure 14. Prompt for unchanged main class

3.4 Discussion

In summary, possible solutions to support a learner to connect different program parts into one are: provide an overview of the program using standard coding guidelines; restrict the order of completion of the program; enable completion of the program in any order after a number of programs but allow the learner to enable completion in a restricted order; and embed default code as expert guidance.

Possible solutions to support a learner in debugging a program are: prompt the learner of a syntax-error as soon as it occurs; and provide some expert guidance in completion of program parts.

Possible solutions to address the small screen and small keypad are: providing default code that minimizes typing; enable completion of the program one part at a time while able to view the full program; and use navigation tabs that allow scrolling across instead of downward.

4. Conclusion and future work

This paper has illustrated how a scaffolding framework has been used to select scaffolding strategies to address learner challenges. A similar approach is applied to all the other challenges that have not been presented. More specifically, the paper has followed a learner-centered methodology where the learners' needs influenced the choice of scaffolding strategies. Also, the paper illustrates how the scaffolding strategies have been implemented on a mobile phone, considering its screen size and keypad limitations, to scaffold construction of Java programs. Therefore, this paper has concretely shown a theoretic derivation of scaffolding strategies, and consequently their implementation on a mobile device.

The use of the scaffolding framework has resulted in the choice of specific scaffolding strategies such as: providing a visual representation of a

Java program by showing an overview of the program parts; enabling interaction with these parts using collapsible and expandable buttons and clickable parts; providing some default code; providing one step navigation ability between the pages; enabling completion of the program one part at a time while being able to view the full program; and providing error prompts as soon as a learner makes a mistake. These scaffolding strategies address the cited learners' needs and also limitations of mobile phones such as small screen size and small and soft keypad.

Current and future work consists of testing the application with learners of programming in different African universities. The evaluation of the results from the experiments seeks to understand two issues: which of the theoretically derived scaffolding strategies are appropriate, and which are inappropriate, to support construction of programs on a mobile device; and how learners use the scaffolds as they construct programs on a mobile device.

Acknowledgement

This work is supported by the Hasso Plattner Institute, and the ICT for Development laboratory at University of Cape Town.

References

Ackermann, E., 1996. Perspective-Taking and Object Construction. In Y..a.R.M. Kafai, ed. *In Constuctionism in Practice: Designing, Thinking, and Learning in a Digital World*. Mahwah, New Jersey: Lawrence Erlbaum Associates. pp.Part 1, Chap.2.

Adipat, B. & Zhang, D., 2005. Interface Design for Mobile Applications. In *Proceedings of the Eleventh Americas Conference on Information Systems*., 2005.

Apiola, M., Tedre, M. & Oroma, J.O., 2011. Improving Programming Education in Tanzania: Teachers' and Students' Perceptions. In *Proceedings 41st ASEE/IEEE Frontiers in Education Conference*., 2011.

Bentley, T., 2012. *Learning Beyond the Classroom*. Taylor & Francis.

Churchill, D. & Hedberg, J., 2008. Learning object design considerations for small-screen handheld devices. *Computers and Education* , 50, pp.881-93.

Elias, T., 2011. Universal instructional design principles for mobile learning. *The International Review of Research in Open and Distance Learning* , 12(2), pp.143-56.

Guzdial, M. et al., 1998. upporting Programming and Learning-to-Program with an Integrated CAD and Scaffolding Workbench. *Interactive Learning Environments*, 6(1-2), pp.143-79.

Hristova, M., Misra, A., Rutter, M. & Mercuri, R., 2003. Identifying and correcting Java programming errors for introductory computer science students. In *SIGCSE '03 Proceedings of the 34th SIGCSE technical symposium on Computer science education*., 2003.

Jackson, S., Krajcik, J. & Soloway, E., 1998. The design of guided learner-adaptable scaffolding in interactive learning environments. In *CHI '98 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Los Angeles, CA USA, 1998.

Jones, M. et al., 1999. Improving Web interaction on small displays. *Computer Networks*, 31(11), pp.1129-37.

Kukulska-Hulme, A., 2005. Mobile usability and user experience. In A. Kukulska-Hulme & J. Traxler, eds. *Mobile Learning*. Taylor & Francis. p.47.

Kukulska-Hulme, A., 2007. Mobile Usability in Educational Contexts: What have we learnt? *International Review of Research in Open and Distance Learning*, 8(2).

Luchini, K. et al., 2002. Scaffolding in the small: designing educational supports for concept mapping on handheld computers. In *CHI'02 Extended Abstracts on Human Factors in Computing Systems*., 2002.

Luchini, K., Quintana, C. & Soloway, E., 2004. Design Guidelines for Learner-Centered Handheld Tools. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. New York, 2004.

Maleko, M., Hamilton, M. & D'Souza, D., 2012. Novices' perceptions and experiences of a mobile social learning environment for learning of programming. In *ITiCSE '12 Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*., 2012.

Quintana, C. et al., 2009. A Scaffolding Design Framework for Software to Support Science Inquiry. *Journal of the Learning Sciences*, 13(3), pp.337-86.

Sivasakthi, M. & Rajendran, R., 2011. Learning difficulties of 'object-oriented programming paradigm using Java': students' perspective. *Indian Journal of Science and Technology*, 4(8), pp.983-85.

Sun-Microsystems, 1997. *Code Conventions for the Java Programming Language*. [Online] Available at: http://www.oracle.com/technetwork/java/codeconventions-150003.pdf [Accessed 19 November 2013].

Tillmann, N. et al., 2012. The Future of Teaching Programming is on Mobile Devices. In *ITiCSE'12 Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*. Haifa, Israel, 2012.

Vihavainen, A., Vikberg, T., Luukkainen, M. & Pärtel, M., 2013. Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*., 2013.