2013

Distractions in programming evironments

Raina Mason
Southern Cross University

Graham Cooper
Southern Cross University

Publication details

Mason, R & Cooper G 2013, 'Distractions in programming evironments', in A Corbone & J Whalley (eds), *Proceedings Fifteenth Australasian Computing Education Conference (ACE2013)*, Adelaide, SA, 29 January - 1 February, Australian Computer Society series; 136, pp. 23-30.

Distractions in Programming Environments

Raina Mason
Southern Cross University
raina.mason@scu.edu.au

Graham Cooper
Southern Cross University
graham.cooper@scu.edu.au

Abstract

A workshop for teaching introductory programming using Lego Mindstorms NXT presented students with either a 'complete' or 'subset' form of user interface, both of which are pre-packaged with the application. The learning activities presented to all students only made use of the functionality contained within the subset interface and students presented with the complete interface only made use of the functionality associated with the subset version. Despite no reference to, or use being made of, the extended functionality of the complete interface, students undertaking activities in this mode reported higher levels of difficulty associated with learning programming and performed poorly on a test of programming concepts compared to students presented with the subset form of the interface. Results are explained in terms of Cognitive Load Theory, in particular, redundancy of information. Implications to the selection of programming languages and environments for teaching introductory programming are discussed.

Keywords: Cognitive load, instructional design, introductory programming, Lego Mindstorms.

1 Introduction

Teaching programming is difficult and often unsuccessful (McCracken et al., 2001; Denning and McGettrick, 2005; Ma et al., 2007). This difficulty may be partially caused by the necessity of learning several interacting concepts all at the same time, such as understanding the problem statement, constructing algorithms, navigating syntax rules, semantics, problem solving, navigating a programming interface and compiling and executing a program (Jenkins, 2002).

Regardless of the programming language being used or other aspects of learning programming such as the mechanics of compiling, students must learn how to construct algorithms using the three core structures of sequence, selection and iteration (Dijkstra, 1972). The primary conduit for teaching and learning these three core structures is the user interface, environment and programming language.

Introductory programming courses have many languages and environments from which to choose. There has been debate (Kolling, 1999; Kelleher and Pausch, 2005) regarding the extent to which simplified learning environments may be beneficial in this context. Some have argued, and developed simplified environments, for pedagogical reasons, but little direct evidence for the reasons behind their success in this purpose have been demonstrated. In contrast, some introductory programming course instructors have contended (see Mason et al. 2012) that introductory programming courses should utilise "professional" level applications as these are used in industry, and that there is no benefit to be gained in requiring students to learn an additional language and environment as a pathway to such professional languages and environments.

The current paper argues that the relatively high level of complexity inherent within most programming languages and environments acts as an impediment to understanding and learning the three core structures of sequence, selection and iteration. Although arguments have been offered for the effectiveness of simpler environments based upon student motivation, visual aspects of the interface and games oriented tasks (Kolling and Henriksen, 2005; Boisvert, 2006; Gomes and Mendes, 2007; Hundhausen et al., 2009) such environments have not been analysed through the lens of cognitive learning theories.

Theories of cognition such as Cognitive Load Theory (Sweller 1999) have been used to develop successful principles of instructional design in other complex domains (for example, see Van Merrienboer et al., 2006). Cognitive Load Theory has previously demonstrated the negative impact that can accrue from split attention (Tarmizi and Sweller, 1988) and redundant information (Mayer et al., 2001). Such studies have typically required students to attend to the 'complete' set of information presented. In the context of learning programming there are elements of information that are presented on screen – within the environment – that are irrelevant to the task(s) at hand. Even though such elements may reside outside of the task activities undertaken by students, this paper argues that these may effectively become a source of split attention or redundancy through tacit distraction and thus impede learning.

The primary purpose of the current research is to explore the potential benefits of using simplified languages and environments as pathways to more positive learning experiences, increased self-efficacy with respect to programming, and deeper understanding and transfer of acquired concepts to other computer programming languages.

2 Methodology

2.1 Participants

The experiment was conducted as part of an "IT Careers Day" at a private high school in regional Queensland, with 32 students in Year 7, aged 11 to 13 years. Students participating in the day completed an introductory programming workshop with Lego Mindstorms NXT robotics (The LEGO Group 2009) followed by a Mindstorms workshop test, then attended a normal school period of class-work. After lunch they participated in an IT careers information session followed by another introductory programming workshop involving the use of the Alice programming environment (Carnegie Mellon University 2006).

2.2 Mindstorms Robots

The Lego Mindstorms NXT kits enable students to build and program robots of their own design. Each kit consists of a central controller "brick" with processing hardware and software, touch, sound, light and ultrasonic sensors, and several servo motors, coupled with 'technics'-style Lego pieces and gears to enable building of several forms of robots. These robots can respond to input through the sensors and can show output by sound, visual display and movement. The robots are programmed using the Mindstorms NXT software, using a USB cable connection to a PC.

The Mindstorms NXT programming environment is highly visual. Programs are created by dragging programming "blocks" – which are presented as icons representing their functionality - to a timeline, and then setting properties of each block by typing in values or selecting options. The programs are executed in sequence along the timeline. More complex programs can be constructed by using loop and switch/decision structures, as well as 'wait' blocks which can be likened to event handling in more traditional languages.

2.3 Design of Workshops

There are two modes of interface presentation available for the Mindstorms software. A "subset" version of the interface presents a truncated set of icon blocks, which are sufficient to build many programs, but lack functionality for tasks such as data storage, data retrieval, mathematical calculations and more specialized functionality. A more "complete" version of the interface presents a greater number of icon blocks representing a greater range of programming tasks, and these are organized into a menu-submenu format.

It was hypothesised that students given either interface would experience an increased knowledge of IT, increased intent to pursue IT as a career, decreased perceived difficulty of programming and increased self-efficacy in programming. The students presented with the subset version of the interface, however, were hypothesised to experience and report amplified effects, resulting in higher self-efficacy and lower levels of the perceived difficulty of programming, than those presented with the more complete interface. It was further hypothesized that students presented with the subset interface would outperform those who received the complete interface on subsequent knowledge tests as measured by both time and score. It was also hypothesised that students who received the subset interface would be more able to transfer their newly acquired (general) knowledge and skills in computer programming to the Alice computer programming environment.

For these reasons, each student was asked about their knowledge and attitudes towards IT and programming before the IT careers day and at the end of the day, after the Alice workshop. Each student was also asked about the conscious cognitive load which they experienced during the workshop and was given a performance test directly after the Mindstorms workshop. The sequence of activities is presented in Figure 1.

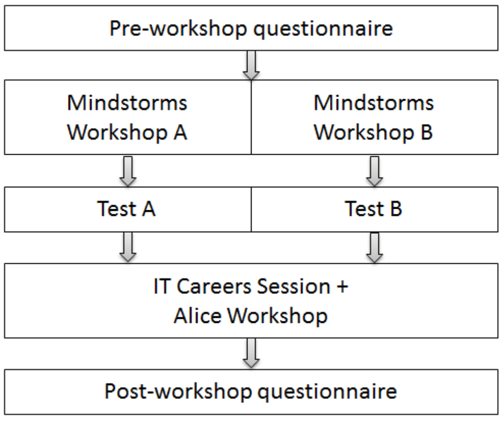
Pre-workshop questionnaire
Mindstorms Workshop A
Mindstorms Workshop B
Test A
Test B
IT Careers Session + Alice Workshop
Post-workshop questionnaire

Figure 1: Sequence of Activities

2.4 Instructional Design

Each workshop involved students working either alone or in pairs at one PC computer with one shared robot (built into a humanoid form). There were the same number of pairs and single working students in each group. The instructor introduced the participants to the physical robots and the Mindstorms NXT software and then worked through a series of programming activities with the students. Each programming activity consisted of the instructor demonstrating and describing a small worked example on a large smartboard. The participants then replicated the example using the software, downloaded their solution to the robots and ran their programs. After each student or pair of students completed the activity successfully, the next worked example was demonstrated. At the end of the workshop, students were given time to create their own more complex programs.

The approach used was designed to reduce extraneous cognitive load, and thus facilitate effective learning. Worked examples were used because worked examples have been proven to be a more effective teaching approach than problem solving for novices in technical domains (for examples see Sweller and Cooper, 1985; Cooper and Sweller, 1987; Zhu and Simon, 1987; Paas, 1992). The instructor used verbal explanations at the same time as pictures and processes were demonstrated on the screen, in accordance with the modality principle (Mousavi et al., 1995; Tindall-Ford et al., 1997). The explanations of each step were provided at the same time as the on-screen demonstrations, to reduce cognitive load caused by the split-attention effect (Chandler and Sweller, 1991; Mayer and Anderson, 1991). The instructor also followed a pre-defined script designed using the segmentation principle (Mayer & Chandler 2001) which advocates delivering content in small learner-paced segments of delivery, moving from simple examples to more complex ones.

The sequence of activities included using the programming environment, simple block use and setting of properties, sequence, looping, events (sensor triggers) and more complex combinations of these concepts. Due to workshop time constraints, decision structures were omitted.

2.5 Treatment Groups

2.5.1 Interfaces

As previously described, the Mindstorms NXT software uses programming blocks to build programs using drag and drop placement on a graphical timeline. The default palette of blocks includes the most commonly used programming blocks and is the first palette available when a new program is created for the first time. Most programming blocks are available on the left of the screen, with one group of programming blocks situated in a slide-out sub-palette. This interface was designated as the Subset interface (see Figure 2).

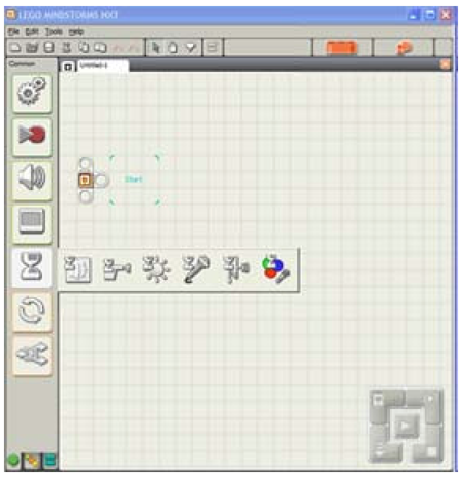

Figure 2: Mindstorms Subset Interface

The software can also be configured so that a more comprehensive set of programming blocks are available when a new program is created. This more comprehensive palette includes the common blocks available in the Subset interface and repeats these blocks and adds new blocks in several sub-palettes accessed through side buttons/icons (for example see Figure 3).

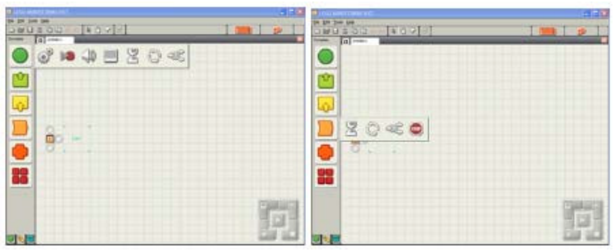

Figure 3: Mindstorms Complete interface

The user has a choice of more programming blocks in this interface, from now referred to as the Complete interface to distinguish it from the Subset interface. The buttons are the same size in each interface and buttons that have the same functionality have the same appearance.

2.5.2 Distractors

Cognitive Load Theory has previously demonstrated that redundant information may interfere with learning (Mayer et al., 2001). In the context of the activities undertaken by students the Complete interface presented icon blocks that were unnecessary to the tasks to be undertaken. These additional icon blocks were never referenced in any instructions or activities. It is hypothesised that their mere presence on screen would act as a form of tacit distractor. It was hypothesised that this would detract from student learning of the interface. It was further hypothesised that this tacit distraction would also reduce participants learning of the core underlying computer programming concepts and procedures being presented within the learning activities. It was hypothesised that this would lead to lower levels of self-efficacy. Finally, and most importantly, it was hypothesised that such tacit distraction would also reduce the scope to transfer newly learnt concepts and procedures to other programming environments.

2.5.3 Groups

Students were divided into two groups/workshops with roughly equal numbers of each gender (Group Subset: 9 males, 8 females; Group Complete: 7 males, 8 females) in each group. Group Subset used the subset interface throughout their workshop. Group Complete used the complete interface throughout their workshop. Both groups used the same programming blocks, and completed the same activities in the same time. Even though Group Complete had a greater number of blocks available via the palette on-screen, they were not directed to use any of these extra blocks, in any of the worked example activities or free programming time. The extra blocks were not referenced or explained in any way.

2.6 Instruments

Prior to the careers day, each participant completed a questionnaire consisting of demographic questions about age and school year, as well as several 9-point Likert scale questions concerning their level of computer literacy, programming experience, knowledge of IT, intent to pursue a career in IT, perception of the difficulty of programming, and confidence with programming.

After the Mindstorms workshop, the participants were given an equivalent test dependent on their treatment group. The timed, written tests were designed to test recall of the purpose of various programming blocks, the building of schema about the interface used, near transfer of programming construct concepts as well as far transfer of concepts such as sequence, looping and events. The test instrument was also used to collect data about the mental effort used in each of the three aspects of completing an activity in the Mindstorms workshop: understanding what needed to be done in each exercise, navigating and using the NXT software, and learning and understanding concepts.

At the conclusion of the careers day, after the Alice workshops, participants completed a post-workshop questionnaire which again asked about their level of knowledge of IT, intent to pursue a career in IT, perception of the difficulty of programming and confidence with programming. Participants were also asked about their level of interest in programming with Mindstorms and with Alice, and how difficult they found each of the programming workshops.

3 Results

3.1 Homogeneity of Groups

There was no significant difference between groups in participant age, computer literacy, programming experience, knowledge of IT and intent to pursue a career in IT between Group Subset and Group Complete, before the workshop. A t-test on age returned no significant difference ($Mean_{Subset} - Mean_{Complete} = -0.08$; $t = -0.3$; $df = 24$; $p = 0.77$, and Mann-Whitney U tests on computing skill, programming experience, knowledge of IT and intent to pursue a career in IT all returned no significant difference between the two groups at the 0.05 level (see Table 1). (It should be noted that an alpha level of $p = 0.05$ is used for all tests reported in this paper)

	U_A	z	p
Computer Literacy	87	0.1	0.46
Prog. Experience	97	-0.62	0.27
IT Knowledge	79	0.26	0.40
Career Intent	83.5	0.03	0.49

Table 1: Homogeneity of Groups

3.2 Career Aspirations and IT Knowledge

Although there were 17 participants in Group Subset and 15 participants in Group Complete for the actual Mindstorms workshops and test, not all participants completed both the pre- and post-workshop questionnaires, as some students did not continue to the Alice workshops due to timetable clashes with other classes. The answers to the pre- and post-workshop questionnaires of the 12 participants in each group who did complete both questionnaires were analysed using the Wilcoxon Signed Ranks test. The median, minimum and maximum scores for each are given in Table 2 below.

It was expected that all participants would be more knowledgeable and more likely to consider a career in IT after the workshops, as a result of the positive experiences, and career information given to them. This was obtained with a significant difference in the perceived IT knowledge of participants after the workshop than before the workshop [Wilcoxon Signed Rank test: $n = 24$, $W = 179$, $N_{s/r} = 21$, $z = 3.1$, $p = 0.001$] and a significant difference in participants' intent to consider a career in IT after the workshop than before the workshop [Wilcoxon Signed Rank test: $n = 24$, $W = 128$, $N_{s/r} = 19$, $z = 2.57$, $p = 0.005$].

	n	median	mode	min	max
Knowledge - pre	24	5	5	1	7
Knowledge - post	24	6	7	2	9
Career - pre	24	4.5	1	1	9
Career - post	24	5.5	7	1	9

Table 2: Results of IT knowledge/Career intent

3.3 Programming Difficulty and Self-Efficacy

In both questionnaires, participants were asked to indicate how much they agreed with the statement "I think programming generally is difficult" on a 9 point Likert scale (where 1 = 'no way' and 9 = 'totally!'). It was anticipated that participants would display a decrease in the perceived level of difficulty of programming after the Careers Day when compared to before the Careers Day. A significant decrease was obtained in participants' perception of the difficulty of programming, from before the workshops to after the workshops [Wilcoxon Signed Rank test: $n = 24$, $W = 128$, $N_{s/r} = 19$, $z = 2.57$, $p = 0.005$]. The median, minimum and maximum scores for perceived level of difficulty of programming are shown in Table 3 below.

	n	median	mode	min	max
Pre-workshop	24	5	5	1	9
Post-workshop	24	4	5	1	7

Table 3: Measures of Difficulty of Programming

It was also hypothesised that Group Subset may have a greater shift in perception of difficulty in the direction of "easier" than Group Complete, who were presented with the fuller interface in the Mindstorms workshops.

Further analysis of the pre- and post-workshop answers from the 12 participants in Group Subset and 12 participants in Group Complete who completed the Mindstorms workshops and the Alice workshops and who completed both questionnaires were then analysed individually using Wilcoxon Signed Rank tests. The results are shown in Table 2.

	n	W	$N_{s/r}$	z	p
Subset	12	-55	11	-2.42	0.008
Complete	12	-29	10	-1.45	0.07

Table 4: Difficulty of Programming

Group Subset had a significantly different perception of the difficulty of programming after the workshop compared to before the workshop, in the direction of 'easier'. Although Group Complete displayed shifts downwards, this group did not experience a significant change in their perception of the difficulty of programming.

In pre- and post-workshop questionnaires, participants were asked to indicate how much they agreed with the statement "I feel confident with programming" on a 9 point Likert scale (where 1 = 'no way' and 9 = 'totally!'). Previous workshop results (Mason et al., 2011a, 2011b) indicated that it was likely that participants would report increased self-efficacy in computer programming after the workshops. However it was anticipated that Group Subset may have a greater increase in self-efficacy in programming than Group Complete.

Answers from both groups were analysed together using Wilcoxon Signed Rank tests and participant's self-efficacy was found to be significantly higher after the workshops than before the workshops [n = 24, W = 119, $N_{s/r} = 18$, z = 2.58, p = 0.005]. The median, minimum and maximum scores for participants' self-efficacy before and after the workshops are shown below in Table 5.

	n	median	mode	min	max
Pre-workshop	24	5	5	1	9
Post-workshop	24	7	9	2	9

Table 5: Measures of self-efficacy

Pre- and post-workshop measures were then analysed separately for the subset and complete interface groups using Wilcoxon Signed Rank tests and the results are shown in Table 6 below:

	n	W	$N_{s/r}$	p
Subset	12	42	9	p < 0.01
Complete	12	18	9	p > 0.05

Table 6: Self-Efficacy in Programming

Group Subset had a significantly higher self-efficacy in programming after the workshop than before the workshop. Although Group Complete displayed some shifts upwards in self-efficacy, this group did not experience a significant change.

3.4 Test Performance

3.4.1 Test Completion Time

It was hypothesised that if the different interface had an effect on learning then participants from Group Subset would take less time to complete the test than participants in Group Complete. The times from the 17 participants in Group Subset and 15 participants in Group Complete were compared using a t-test. Participants from Group Subset were found to take significantly less time to complete the test than Group Complete [$\text{Mean}_{\text{Subset}} - \text{Mean}_{\text{Complete}} = -150$ seconds; t = -1.91, df = 30, p = 0.03].

3.4.2 Test Score

There was a total maximum possible mark of 17 for the Mindstorms test. The sequence of questions and associated marks were allocated as follows:

- 1 mark each for correct description of the purpose of programming blocks [total 3 marks];
- 1 mark each for correct placement of programming block on blanked interface [total 6 marks];
- 1 mark each for correct multiple choice answers (near transfer) [total 5 marks];
- 1 mark each for far transfer answers [total 3 marks].

The two groups mean total marks, standard deviation, minimum and maximum are shown in Table 7.

	mean	st dev	min	max
Subset	11.03	2.60	6.5	15
Complete	8.83	1.88	6	14

Table 7: Test Score Totals

The total test scores from the 17 participants in Group Subset and 15 participants in Group Complete were compared using a t-test, and participants from Group Subset were found to have a significantly higher test score than Group Complete [$\text{Mean}_{\text{Subset}} - \text{Mean}_{\text{Complete}} = 2.20$; t = 2.71, df = 30, p = 0.006].

3.4.3 Interface Schema Acquisition

Students were asked to indicate where six programming blocks were placed on a blanked version of the Mindstorms interface. Each of these six programming blocks had been used in the Mindstorms workshop activities at least twice. It was anticipated that students in Group Subset would more effectively build schema for the positioning of the blocks in the interface over students in Group Complete interface, who would have more of their working memory capacity used in needing to deal with the tacit distractors of the availability of extra blocks – even though those blocks were not being used or referenced in any of the activities.

The maximum score available on this question was 6, if the student placed all of the blocks in the correct positions. The Complete Interface makes several programming blocks available in more than one location, so if a student in Group Complete placed that particular block in any of the correct positions, they received a point for that block.

The two groups mean marks, standard deviation, minimum and maximum are shown in Table 8.

	mean	st dev	min	max
Subset	2.76	1.71	0	5
Complete	1.33	1.18	0	4

Table 8: Interface Schema Score totals

Participants from Group Subset scored significantly higher on rebuilding the interface than participants from Group Complete [t-test: $\text{Mean}_{\text{Subset}} - \text{Mean}_{\text{Complete}} = 1.43$, t = 2.72, df = 30, p = 0.005].

3.4.4 Knowledge Acquisition

The maximum score available for the test, *excluding* rebuilding the interface, was 11. The two group scores for the knowledge acquisition part of the test were compared using a t test and although these results were not statistically significant [$\text{Mean}_{\text{Subset}} - \text{Mean}_{\text{Complete}} = 0.7647$, t = +1.36, df = 30, p = 0.09], the results trended in the expected direction. Closer inspection of the test scores for Knowledge Acquisition showed that 9 of the 17 participants in Group Subset scored 9 or higher (from a possible 11), while in Group Complete, only 2 of the 15 participants scored 9 or above. This result is statistically significant (p = 0.02) using Chi-Square analysis. This indicates that more participants in Group Subset scored highly (at 9 or more marks out of a possible 11 marks) for Knowledge Acquisition than those participants in Group Complete.

3.5 Difficulty of Programming Environments

3.5.1 Mindstorms environment Difficulty

In the post-IT Careers Day questionnaire, participants were asked to complete the statement "Programming robots to do things is .. " on a 9 point Likert scale (where 1 = 'really easy' and 9 = 'really difficult'. It was expected that if having the extra (unused) programming blocks available in the Complete interface were having an effect on working memory load while using the interface, then Group Complete would perceive programming robots as more difficult than Group Subset, even though both groups completed the same activities with the same programming blocks.

The responses of the 15 participants in Group Subset and 14 participants in Group Complete who completed the Mindstorms workshops and completed the post-workshop questionnaire were analysed for differences using a Mann-Whitney U Test. Participants in Group Complete found programming with the Mindstorms Robots significantly more difficult than participants in Group Subset [U_A = 144, z = 1.66, p = 0.049]. The median, minimum and maximum scores are given below in Table 9.

	n	median	mode	min	max
Group Subset	15	2	1	1	5
Group Complete	14	3.5	1	1	9

Table 9: Mindstorms Difficulty

This was an expected result. The presence of the extra (unused) programming blocks was interfering with the attentional process for the students in Group Complete.

3.5.2 Alice Environment Difficulty

In the post-IT Careers Day questionnaire, participants were also asked to complete a similar statement about the difficulty of programming with Alice - "Programming with Alice 3D worlds is .. " on a 9 point Likert scale (where 1 = 'really easy' and 9 = 'really difficult'.

Note that both Group Subset and Group Complete were mixed in a common Alice programming session in the afternoon, after completing the Mindstorms workshops.

Both groups completed the same Alice workshop activities, with the same Alice interface, and were asked afterwards about the difficulty of programming in Alice. As all participants were in the same Alice programming workshop, the only variable was the Mindstorms group in which each participant had participated. It was hypothesised that the difference in difficulty experienced in the Mindstorms workshops as a result of having the subset interface for participants in Group Subset, compared to those in Group Complete, would have a positive transfer effect to the perceived difficulty of Alice programming. That is, participants that had experienced the Subset interface for Mindstorms, would perceive programming in Alice as less difficult than those who had experienced the Complete Mindstorms Interface.

The responses of the 15 participants in Group Subset and 14 participants in Group Complete who completed the Mindstorms workshops and completed the post-questionnaire after the Alice workshop were analysed for differences using a Mann-Whitney U Test. The novice programming participants in Group Complete found programming with ***Alice*** significantly more difficult than novice programming participants in Group Subset [U_A = 158, z = 2.29, p = 0.01]. The median, minimum and maximum scores are given below in Table 10.

	n	median	mode	min	max
Group Subset	15	1.5	1	1	7
Group Complete	14	5	5	1	9

Table 10: Alice Difficulty

This result shows a transfer effect from the Mindstorms workshop to the Alice workshop based upon the nature of the Mindstorms interface that participants had received. This is discussed more in Part 4: Discussion.

3.5.3 Cognitive Load

It was expected that participants in Group Subset would report lower conscious mental effort (cognitive load) measures in "navigating and using the Mindstorms NXT software" and their overall patterns of mental effort would differ from participants in Group Complete. The performance measures indicate that learning was more effective for Group Subset. We had hypothesized that this would be the result of lower cognitive load being imposed upon Group Subset. Although there were trends present in the direction of lower reported cognitive load in Group Subset, there were no significant differences between groups.

4 Discussion

The benefits of the Careers Day intervention for both girls and boys who were novices to programming were obvious, with positive shifts in IT knowledge, IT career aspirations and self-efficacy in programming, and lowered perception of the difficulty of programming, from before the workshops to after the workshops.

The effect of the treatment and differences between groups became evident on analysing their performance in the Mindstorms test. On each of the four chosen measures (Test Completion Time, Total Test Score, Interface Schema Acquisition Score and Knowledge Acquisition Score), Group Subset outperformed Group Complete. There was a demonstrated advantage for the students who were given the simpler interface with fewer options, even though the extra options for Group Complete were not used.

It was expected that if the students in Group Complete were experiencing higher cognitive load on working memory during the Mindstorms workshop than Group

Subset as a result of the extra options on screen, they would experience the task of programming in the Mindstorms environment as more difficult, compared to Group Subset. Group Complete did find programming in the Mindstorms environment significantly more difficult than Group Subset which supports this hypothesis.

There were, however, no significant differences observed between groups on reported cognitive load measures. This may be due to lack of sensitivity in the Likert test questions used with this participant pool, or may represent another dynamic associated with a form of tacit distraction. This issue warrants further research.

Both treatment groups then participated in the same Alice workshop, at the same time, with the same materials and instruction. There was a significant transfer effect, with those participants in Mindstorms Group Subset finding Alice programming significantly easier than participants who had been in Group Complete.

This is a potentially critical outcome. At the heart of all computer programming languages and environments are the core, fundamental concept blocks that enable the design and development of suitable algorithms. The results obtained in the present study argue that the schemas for these underlying mental representations may be facilitated through the use of entry-level computer programming environments specifically designed to cater for novices.

These results have implications for the often-debated question about whether it is better to introduce introductory programming students to an Integrated Development Environment (IDE) that is used in industry first, or to introduce students to programming using a teaching environment first, and then move on to an industry standard IDE later. Instructors that are in favour of using an industry standard environment for introductory programming courses often point to the extra effort of teaching (and for students, in learning) two environments (de Raadt et al., 2002; Mason et al., 2012).

The results of this research indicate that for novices to programming, having extra options available in the environment - ***even if they are not used or referenced*** - hinders learning (reduces performance) and causes the students to perceive programming in both that environment and subsequent environments as more difficult. The results of this study indicate that novice students benefited from a simplified first-programming environment. This facilitated learning of core programming constructs as measured on test performance and also transferred to a second programming environment as measured by reported perceptions of programming difficulty.

While the current study was specifically focussed upon novice programmers and programming environments, it is worth noting that relatively many computer applications present users with additional icons and functionality that are redundant (or unnecessary) to their task performance. Such “over-provision” of functionality may be misguided and impede learning of the application.

The current study may have limited scope because it used participants who were school students rather than university students, because the programming environments used were heavily icon based rather than line code, and because the entire exposure of participants was across a single day, rather than across an entire year (or longer). Nevertheless, the design of the specific instructional materials used, and the broad complex of results obtained, were driven by the application of Cognitive Load Theory to the context of teaching novices some of the basic concepts, structures and processes of computer programming.

Cognitive Load Theory provides an extensive body of empirical studies demonstrating utility in enhancing instructional design in complex areas of learning such as mathematics, science and industry technical applications (Sweller, 1999). The results reported here represent another example of Cognitive Load Theory being usefully applied to enhance the design of instructional materials in an area of high conceptual and task complexity…that of computer programming…and all computer programmers begin their life as programmers…as novices.

5 Thanks

The authors would like to acknowledge the helpfulness and involvement of the school teachers and students who were the participants in this study.

6 References

Boisvert, C., 2006. Web animation to communicate iterative development. ACM SIGCSE Bulletin 38, 173–177.

Carnegie Mellon University, 2006. What is Alice and what is it good for? [WWW Document]. URL http://www.alice.org/index.php?page=what_is_alice/what_is_alice

Chandler, P., Sweller, J., 1991. Cognitive Load Theory and the Format of Instruction. Cognition and Instruction 8, 293–332.

Cooper, G., Sweller, J., 1987. Effects of schema acquisition and rule automation on mathematical problem-solving transfer. Journal of Educational Psychology 79, 347–362.

de Raadt, M., Watson, R., Toleman, M., 2002. Language trends in introductory programming courses [WWW Document]. Informing Science + IT Education Conference. URL http://proceedings.informingscience.org/IS2002Proceedings/papers/deRaa136Langu.pdf

Denning, P., McGettrick, A., 2005. Recentering Computer Science. Communications of the ACM 48, 15–19.

Dijkstra, E.W., 1972. Notes on Structured Programming, in: Structured Programming. Academic Press, New York, NY, pp. 1–82.

Gomes, A., Mendes, A.J., 2007. An environment to improve programming education, in: Proceedings of the 2007 International Conference on Computer Systems and Technologies - CompSysTech ’07. ACM Press, Bulgaria, pp. Article 88, 6 pages.

Hundhausen, C.D., Farley, S.F., Brown, J.L., 2009. Can direct manipulation lower the barriers to computer programming and promote transfer of training? ACM

Transactions on Computer-Human Interaction 16, 1–40.

Jenkins, T., 2002. On the difficulty of learning to program, in: Proceedings of the 3rd Annual Conference of the LTSN-ICS. Loughborough, Ireland, pp. 53–58.

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

Kolling, M., 1999. The problem of teaching object-oriented programming, part 2: Environments. Journal of Object-Oriented Programming 11, 6–12.

Kolling, M., Henriksen, P., 2005. Game programming in introductory courses with direct state manipulation, in: Proceedings of ITiSCE'05. ACM Press, Caparica, Portugal, pp. 59–63.

Ma, L., Ferguson, J., Roper, M., Wood, M., 2007. Investigating the viability of mental models held by novice programmers, in: Proceedings of the 38th Technical Symposium on Computer Science Education. ACM Press, pp. 499–503.

Mason, R., Cooper, G., Comber, T., 2011a. Girls get IT. ACM Inroads 2, 71–77.

Mason, R., Cooper, G., Comber, T., 2011b. It's (no longer) a remote chance for girls in IT, in: Proceedings of the 1st International Australasian Conference on Enabling Access to Higher Education 2011. University of South Australia, Adelaide, Australia, pp. 310–321.

Mason, R., Cooper, G., de Raadt, M., 2012. Trends in Introductory Programming Courses in Australian Universities – Languages, Environments and Pedagogy, in: de Raadt, M., Carbone, A. (Eds.), Proceedings of the Fourteenth Australasian Computing Education Conference (ACE2012). Australian Computer Society, Inc., Melbourne, Australia, pp. 33–42.

Mayer, R.E., Anderson, R.B., 1991. Animations need narrations: An experimental test of a dual-coding hypothesis. Journal of Educational Psychology 83, 484–490.

Mayer, R.E., Chandler, P., 2001. When learning is just a click away: Does simple user interaction foster deeper understanding of multimedia messages? Journal of Educational Psychology 93, 390–397.

Mayer, R.E., Heiser, J., Lonn, S., 2001. Cognitive constraints on multimedia learning: When presenting more material results in less understanding. Journal of Educational Psychology 93, 187–198.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B.-D., Laxer, C., Thomas, L., Utting, I., Wilusz, T., 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. ACM SIGCSE Bulletin 33, 125–180.

Mousavi, S.Y., Low, R., Sweller, J., 1995. Reducing cognitive load by mixing auditory and visual presentation modes. Journal of Educational Psychology 87, 319–334.

Paas, F., 1992. Training strategies for attaining transfer of problem-solving skills in statistics: A cognitive-load approach. Journal of Educational Psychology 84, 429–434.

Sweller, J., 1999. Instructional design in technical areas. The Australian Council for Educational Research Ltd, Camberwell, VIC.

Sweller, J., Cooper, G., 1985. The Use of Worked Examples as a Substitute for Problem Solving in Learning Algebra. Cognition and Instruction 2, 59–89.

Tarmizi, R.A., Sweller, J., 1988. Guidance during mathematical problem solving. Journal of Educational Psychology 80, 424–436.

The LEGO Group, 2009. MINDSTORMS [WWW Document]. URL http://mindstorms.lego.com/en-us/Default.aspx

Tindall-Ford, S., Chandler, P., Sweller, J., 1997. When two sensory modes are better than one. Journal of Experimental Psychology: Applied 3, 257–287.

Van Merrienboer, J.J.G., Kester, L., Paas, F., 2006. Teaching complex rather than simple tasks: balancing intrinsic and germane load to enhance transfer of learning. Applied Cognitive Psychology 20, 343–352.

Zhu, X., Simon, H.A., 1987. Learning mathematics from examples and by doing. Cognition and Instruction 4, 137–166.