Learning challenges faced by novice programming students studying high level and low feedback concepts

Matthew Butler and Michael Morgan
Faculty of Information Technology
Monash University

This paper describes an investigation into the nature of the academic problems that face novice programming students. These learners are required to demonstrate competencies in high-level abstract principles of programming and logic, such as program design and OOP principles, which are conceptually difficult. During the programming task learners receive relatively high levels of feedback on low level issues, such as syntax rules, but tend to receive low levels of feedback on conceptually more difficult issues. This problem tends to be exacerbated by the trend of learners to study independently, outside the classroom or in online modes, which further reduces the options available for quality feedback on high-level issues. This paper analyses the results of a survey given to students enrolled in an introductory programming unit across three campuses at Monash University in 2007. The survey focused on student perceptions of the relative difficulty in understanding and implementing both low level-programming concepts, such as syntax and variables, and high-level concepts, such as OOP principles and efficient program design. An analysis of the approximately 150 responses has indicated that a significant percentage of students experienced difficulties in high-level concepts. Also while many students may indicate an understanding of the principles of many high level concepts more students reported experiencing difficulty in implementing such concepts. This indicates that many students may achieve a level of understanding allowing near transfer of domain knowledge but fail to reach a level of understanding that enables far transfer.

Keywords: programming curriculum, novice programmers, feedback

Introduction

In order for curriculum designers and educators to ensure that new Information Technology students can make as smooth a transition as possible into their new discipline, it is important to understand the core problems that students studying introductory programming encounter. Traditionally first year introductory programming courses have a relatively high fail rate. Hagan et.al (1997) highlighted this concern 10 years ago, indicating that "programming was considered the most difficult and least interesting subject by most first year students in all Computing courses at Monash University" (pg. 37). Bennedsen and Caspersen (2007) sought to validate this concern with a survey of universities and colleges worldwide. Although they could not confirm that failure rates were abnormally high in programming units, they did find that pass rates were on average of the order of only 67%. The causes of such high fail rates may be related to a number of factors. This paper focuses on three main elements:

1. The conceptual difficulty of various elements of the curriculum,
2. The level of feedback that is available to students with regard to various components of the programming task,
3. How patterns of study, namely low levels of face-to-face contact experienced by independent learners, impact on the first two issues.

The hypothesis that this study seeks to examine is that student experience conceptual difficulties with elements of the curriculum that require abstract and logical thinking but it is precisely these elements of the curriculum for which little feedback is available, therefore student performance in these areas is poor. Additionally it is hypothesised that this poor performance may be exacerbated by independent study patterns.

Background

First year Information Technology students face a wide variety of challenges. Not only must they contend with the pressures of commencing tertiary education, with all the issues associated with adjusting from secondary to university study, they also are confronted with immersing themselves into a discipline in which they may not have had any prior formal education and for which they must essentially learn a new language, a programming language. For many Information Technology courses, a rudimentary background in mathematics and English is all that is required to enter the degree and commence study. However a first semester of study may encompass such units as Computer Systems, Data Communications and Computer Programming. Students may find themselves in unfamiliar territory and these computing fundamentals can prove to be a learning challenge, particularly introductory programming courses.

Introductory computer programming has been the subject of many research papers, focusing on a wide range of technical and educational aspects. Giangrande (2007) highlights that the issues include "which programming language should be used, which methodology should be taught, which topics should be included" (pg. 153). Debates over Structured versus Object-Oriented driven curriculum still continue to divide computing educators. As Lister et. al. (2006) discuss, "The SIGCSE community is currently sustaining a very vigorous debate on the teaching of programming, with particular regard to the question of objects first" (pg. 147). Research by Schulte and Bennedsen (2006) showed that 79% of surveyed universities covered Object Oriented concepts, with 52%, slightly over half, covering objects first (pg. 20). The debate also rages as to the appropriate choice of programming language, with many favouring widely used languages such as C++ and Java while others advocate 'conceptual' languages or alternate approaches such as the use of games or toolkits in order to focus on logical thinking and implementation. Schulte and Bennedsen (2006) showed that Java is clearly the most used programming language, in the order of 52% across the universities surveyed (pg. 20). Even though this may be the case, many researchers argue against the use of Java. Hadjerrouit (1998) provides a critical evaluation, highlighting the inherent difficulties in using Java as a first programming language, Crawford and Boese (2005) suggest the use of the multimedia language ActionScript as a solution, while others (Powers et. al. 2006; Gross and Powers 2005) choose independent toolsets as a means of introducing students to programming concepts.

Regardless of what choices are made for programming approach, language and development environment, in the classroom the students will still face a challenging combination of abstract programming concepts and logical reasoning processes. These abstract principles and logical reasoning must be applied to solve a variety of real world problems in a variety of contexts. Therefore rote learning is near impossible in the programming context. Although students can arm themselves with an array of programming examples and constructs, each programming problem will have a unique solution comprised of the programming building blocks they have studied. This applied programming, which can be equated to far transfer in learning theory, is fundamental to successful learning. Bruce (2005) highlights a concern that "weaker students entering CS 1 have a very difficult time dealing with the additional layers of abstraction resulting from the use of objects and design patterns" (pg. 111).

This paper presents results of a survey of student perceptions while studying a first semester introductory programming unit, FIT1002 Computer Programming at Monash University, where students learn the Java programming language. The students enrolled in this unit represent a wide range of sub-disciplines in the Information Technology field and the unit is delivered at a number of the university's campuses. The next section of the paper outlines the context of the study at Monash University and goes on to raise the three main issues under investigation; the conceptual complexity of elements of the curriculum, the levels of feedback available to students, and issues associated with independent learning.

Conceptual complexity in introductory programming

Monash University is an internationally recognised institution based in Melbourne, Australia, and is the largest university in Australia. Its Faculty of Information Technology was the first stand-alone IT faculty in Australia, and is in fact the only 'Group of 8' university to have a dedicated IT faculty. In 2005 the faculty undertook a considerable redesign of its undergraduate courses to cater for a down turn in student demand and shifting industry needs. Consultation with industry stakeholders and the Australian Computer Society (ACS) resulted in a set of core units, common to all undergraduate IT courses. These core units cover fundamental IT knowledge, including introductory programming.

The core unit FIT1002 Computer Programming is one of four subjects that new students undertake in their first semester of study. The aim of the unit is to provide these new students, some of whom have no prior programming experience, with the fundamentals of problem solving, program design and implementation in the context of the Java programming language. As this is an introductory unit, the focus is on programming fundamentals and includes such topics as Algorithm Design, Structured Programming Techniques and Control Structures, Object Oriented Design and Programming, and Testing and Debugging Approaches. These topics are consistent with those shown by Shulte and Bennedsen (2006) in their survey as the topics considered most 'relevant' (pg. 22). In addition, objectives of the unit relate to much broader elements, such as 'adopt a problem-solving approach' to 'design object oriented solutions', objectives that focus on principles rather than relating specifically to the Java programming language. As a result the assessment reflects this diversity, with assignment work covering Algorithm Design (Assignment 1), Structured Programming Implementation (Assignment 2) and Object Oriented Design and Implementation (Assignment 3). The examination for this subject covers the entire range of these topics.

As the unit is considered introductory and covers both basic programming paradigms (structured and object oriented), the curriculum spans from low to high levels of conceptual complexity. It is one of the challenges of teaching programming that the balance must be found between providing support for the low level issues, such as the syntax needed to implement programming functions that allow students to see concrete examples of programs work in action, while still covering highly complex conceptual areas such as object orientation, that allow efficient program design techniques. The level of conceptual difficulty of the subject matter is an important consideration in the delivery of learning materials. In the programming domain issues such as Syntax operate at a low level of conceptual difficulty, as although there is a level of semantic understanding required the knowledge is clearly defined, easily verifiable, and at an introductory level the range of syntax covered is not that large. A loop or decision may be considered as a mid level concept, as while there is indeed a conceptual element to their function, the leap from concept to implementation is not great. However the theory of classes and objects as well as their role in program design can be seen as a high level concept as it operates at a level very far removed from their actual application. Although these higher level concepts can be broken up into appropriate segments for easier understanding, it appears as though the leap from *understanding* these concepts to *applying* them can be very difficult for novice programmers. Lahtinen et. al. (2005) suggest this very issue, that "the biggest problem of novice programmers does not seem to be the understanding of basic concepts but learning to apply them" (pg. 17). Winslow (1996) expands on this idea, highlighting that novices are "limited to a surface knowledge of subject" while experts "have a deep knowledge of their subject which is hierarchical and many layered" (pg. 18).

On reviewing the curriculum set for FIT1002 various components of the course can be assessed in terms of the conceptual difficulty of the subject matter. Table 1 shows the approximate arrangement of the curriculum and lists the notional level of conceptual difficulty of the subject matter expected by the authors of this paper based on the nature of the subject matter involved.

Table 1: FIT1002 Curriculum and notional conceptual difficulty

Curriculum	Conceptual Difficulty
Algorithms	High
Syntax	Low
Variables	Mid
Decisions and Loops	Mid
Arrays	Mid
Methods	High
OO Concepts	High
Overall Program & Object Design	High
Testing and Debugging	Mid

The indicator given to the conceptual difficulty of the curriculum is notional and is based simply on the elements of categorisation raised above, along with anecdotal evidence. The survey given to students will be used to ascertain if this categorisation is correct from the perspective of the students enrolled in FIT1002 Computer Programming.

Feedback in introductory programming

One significant problem in applying high level concepts for novice programmers may be in the levels of feedback that are provided to these students during the programming process. Different types of feedback can be obtained by novice programming students, including:

- Feedback from teaching staff
- Feedback from the development environment
- Feedback from testing

Feedback from teaching staff is fundamental to the success of students at an introductory level. This comes primarily from in class interactions, whereby staff can help guide students through the learning process, critique progress, and answer queries students may have. This feedback can also be offered during consultation sessions outside of class time. Staff feedback also comes in the form of assessment of programming assignment work, where comments on design and code quality can be given along with assessment of functionality.

Feedback from the development environment and testing can be received by the student both in and outside of the teaching environment. The development environment provides very specific feedback on programming syntax and low-level code construction. Students are informed when trying to compile a program if there are syntactic problems, and are provided with basic messages to help in rectifying those problems. The development environment can also provide feedback on concepts such as loops, decisions, and other programming concepts, however this feedback relates primarily to implementation rather than fundamental design. Testing also provides base level feedback, in this case if the program actually performs the expected tasks and produces the required output. By considering suitable test data and expected outcomes, students can see if their programs are functioning correctly.

If this feedback is considered in conjunction with the prior discussion of high and low level concepts then problems begin to emerge. Feedback from the development environment and testing relate only to mid to low level concepts, while in contrast teaching feedback covers high-level concepts. A development environment cannot offer feedback on overall program design, while guidance on object oriented principles is limited solely to syntactic implementation. This is of considerable concern given researchers such as Eckerdal et.al. (2006) highlight that a “fundamental goal of undergraduate computer science programs is that graduates be able to design software systems” (pg. 403)

Study patterns

These issues are amplified when coupled with a distinct change in study pattern of students that results in less face to face teaching contact. Anecdotal and observational evidence suggests that students now spend much less time on campus, preferring to study off-campus in distance education modes. It is clear that there are now increased pressures on students of work and family, meaning that much less time is spent on campus in general. Also, computer hardware and software used to be considerably more expensive, however costs have decreased dramatically in the last decade. As a result many students have hardware that in many cases is superior to that on campus and so have less need to spend time on campus in order to access equipment. These factors generate several concerns with the teaching and learning process. With students spending more time off-campus than ever before, an opportunity for high-level feedback is being significantly compromised. How can students be sure that their programs are appropriate from a design perspective if the only feedback they receive outside the classroom relates to implementation?

Survey design

In order to support the literature and anecdotal evidence relating to these concerns with conceptual complexity, feedback and study patterns, a survey of students studying FIT1002 Computer Programming at Monash University was conducted in May 2007. The unit has approximately 500 students enrolled across 6 campuses.

The survey consisted of a number of quantitative and qualitative questions, split into 5 sections. Section A asked for preliminary information relating to the student, such as age, gender, student type, enrolled undergraduate degree. The main objective of this section of the survey was to find out the degree being studied, as this was expected to provide insight into the differing challenges perceived by students with varying IT focuses.

Section B contained questions relating to the students general aptitude and prior programming experience. As the research focuses on specific challenges relating to novice programmers, it was important to obtain an understanding of the aptitudes and natural inclinations of the students. It was hoped that this would allow the researchers to obtain an insight into the extent to which student aptitude and past experience with programming impacted on their perceived understanding of curriculum.

Section C asked students to comment and reflect on their study patterns for the unit. Simple measures of how much time was spent performing each type of study were collected, along with the students perceptions of what the most (and least) important study modes were and their reasons for making this judgement.

Section D was arguably the most important section, asking students to self-evaluate both their understanding and ability to apply key introductory programming concepts. Students were first asked to rate on a 7-point scale the perceived level of difficulty of understanding of each major topic in the curriculum. In a separate question, over the page, students were then asked to rate using the same scale their perceived level of difficulty in implementing each aspect of the curriculum. Lahtinen et. al. (2005) suggest that "students overestimate their understanding" (pg. 17), therefore such a distinction may guide students into a deeper level of self-assessment. It was also intended that this section would indicate if indeed higher-level concepts such as program design proved to be more challenging to students. An understanding of the distinctions between near and far transfer of knowledge was therefore the aim of this series of questions. It should be noted that curriculum topics were kept fairly generalised so that students did not have to reflect on too many different elements. It is assumed that future data collection will focus on specific parts of the curriculum in greater detail.

The final section of the survey provided the students with an opportunity to comment on overall issues in studying the programming unit (FIT1002). This was intended to capture any thoughts that may not have been covered by the previous questions.

The survey was offered to students at the 3 Melbourne based campuses of Monash University. These cohorts were chosen for several reasons:

- It provided access to approximately two-thirds of the students enrolled in the unit,
- The ability to survey students enrolled in a number of different undergraduate degrees, ranging from Computer Science to Business Systems to Multimedia,
- The ability to offer the survey easily in class, rather than require online or e-mail submission.

As a result, students studying the unit at 3 campuses (Gippsland, Malaysia and South Africa) were not surveyed. This was not seen as problematic, as a diverse range of students could be obtained by focusing on the 3 metropolitan campuses. This diversity encompassed both 'prior skills' and 'focus of study' in a broad range of IT disciplines. Two of the campuses not surveyed were international campuses (located in Malaysia and South Africa) and while these campuses were of definite interest, in terms of the research as they could provide insight into some cultural issues, logistical issues prevented the delivery of the survey to these cohorts at the time. However a significant number of international students were surveyed, so the lack of results from the international campuses was not considered a major issue with this study.

Surveys were administered in the revision lecture for the unit, in final week of the 13-week semester, at each campus. Students received an explanatory statement and consent form, and had complete freedom to give or withhold consent to participate. The survey timing was deliberate so as to hopefully achieve the highest possible response rate, but most importantly to survey the students in a period of self-reflection about the nature of the content delivered in the unit due to the examination revision process.

Data analysis and summary of survey results

Of the total number of students enrolled in the unit at the metropolitan campuses surveyed (379), 173 responses were obtained, giving a response rate of 46%. Of these 173 responses, after the data collation process, 167 surveys (n=167) contained usable data, with only 9 being blank or containing responses that could not be used due to factors such as extreme outliers or inappropriate comments. Of the 167 usable surveys, 27% were fully completed, with the remaining surveys containing some questions that were not answered. For the purpose of initial data analysis these can be counted, as questions not answered were predominantly those of a qualitative nature, whereas questions asking for quantitative responses, such as time spent, ratings, rankings, and the like were generally fully answered. This may be something to

consider for further analysis, whether non-responses to qualitative questions was due to time constraints, the length of the survey, or a difficulty in the self-reflection task.

Initial analysis of data has focused on the quantitative elements of the survey. The survey was designed such that qualitative questions could provide further insight into scores and ratings given by the participants. As many of the quantitative questions require the participants to reflect on their experiences with the programming unit and its curriculum, open ended questions provided the ability to seek clarification as to why certain responses were made. For a preliminary discussion of the survey findings however, many initial conclusions could be found solely using the quantitative data. As the research is focusing on student issues with conceptual complexity, the bulk of discussion will be afforded to this area.

To provide a background to the responses, some preliminary information about the participants can be identified based on responses to the initial questions in the survey. In summary:

- Approximately 35% had no prior programming experience at all.
- Participants indicated that on average they spent 9.75 hours per week studying the unit. It should be noted that students enrolled in this unit were expected to attend 5-6 hours per week in scheduled lectures, laboratories and tutorials. Monash University's minimum expectation of student workload per unit is 12 hours.
- On average students indicated that they spent approximately 1 hour of their time on campus in independent study mode, approximately 2.5 hours off campus in independent study mode and approximately 0.5 hours in peer group study mode.
- Laboratory classes were considered the most important study activity, placed first by 31% of participants. Lectures were second, placed first by 28% of participants. It should be noted that study with peers was rated most important by only 3% of respondents. This order of importance was also reflected when taking into account a weighted analysis of all responses.
- Qualitative responses indicate that laboratory sessions were seen as most important in terms of feedback as they provided an opportunity for participants to seek help. This is important to note in consideration of levels of student feedback for various components of the curriculum.
- Lectures were seen as important in general because they provided the participants with the overview of the curriculum to be covered in each week. This appears slightly at odds with research by the likes of Huet et. al. (2004) who suggest that "students… are often sceptical of the effectiveness of this technique" (pg. 5)

As indicated in the discussion of the survey design however, the self-evaluation of the students with regard to the curriculum of the unit was the key set of questions. It was the intent of this section to determine if indeed students expressed difficulty with curriculum of a more conceptual nature, and if a shift in perceived difficulty occurred in applying their understanding of topics to real programming problems.

With regard to perceived *understanding* of the curriculum, the topic of *Object Oriented Concept* was seen as the most challenging. Table 2 below summarises the average of the responses (between 1 and 7) for each topic:

Table 2: Average difficulty rating for understanding curriculum (n=167)

Topic	Conceptual Difficulty	Average Rating (out of 7)
Algorithms	High	1.98
Syntax	Low	2.56
Variables	Mid	2.14
Decisions and Loops	Mid	2.53
Arrays	Mid	4.17
Methods	High	3.92
OO Concepts	High	4.92
OO Design	High	4.52
Testing	Mid	3.60

Results show that topics that can be considered to be of a more conceptual nature were perceived to be more difficult to understand for novice programming students. Table 1 earlier notionally identified Object Oriented concepts along with Object Oriented design as two areas of high conceptual difficulty. This categorisation is consistent with the evaluation provided by the survey participants. It is also important to note that the second and fourth highest ranked topics related specifically to program design. Program design was identified in section 4 of this paper, Feedback in Introductory Programming, as a programming area that provides little feedback to the student.

Perceived ability to *implement* these elements of the programming curriculum produced a similar set of results, as can be seen in Table 3:

Table 3: Average difficulty rating for implementing curriculum (n=167)

Topic	Conceptual Difficulty	Average Rating (out of 7)
Algorithms	High	2.21
Syntax	Low	2.50
Variables	Mid	2.32
Decisions and Loops	Mid	2.68
Arrays	Mid	4.19
Methods	High	4.10
OO Concepts	High	5.12
OO Design	High	4.60
Testing	Mid	3.87

Again, the highest rated elements of the curriculum were those of a higher conceptual nature. Arrays may be considered the exception, ranked the third highest in both tables of results. This is not considered surprising however, as arrays can be seen as applied loops combined with more detailed processing logic, providing significant challenge for novice programmers.

What can also be seen from the two tables is the increase in difficulty rating for almost all elements of the curriculum when going from understanding to implementation. Table 4 below shows the average changes in score as both a figure out of 7 and as a percentage:

Table 4: Average shift in difficulty rating from understanding to implementation

Topic	Conceptual Difficulty	Average Rating Increase (out of 7)	Percentage Increase
Algorithms	High	0.23	3.36
Syntax	Low	- 0.06	- 0.84
Variables	Mid	0.18	2.64
Decisions and Loops	Mid	0.14	2.05
Arrays	Mid	0.03	0.37
Methods	High	0.18	2.61
OO Concepts	High	0.20	2.80
OO Design	High	0.09	1.24
Testing	Mid	0.27	3.92

The percentage increases are only minor, however are consistent across all topics aside from syntax. What is important to note is that this is also consistent with earlier discussion relating to levels of feedback afforded to certain parts of the programming process. Program design was acknowledged as an aspect providing little feedback and problems inherent in this are indicated by both the difficulty ratings and the increase in rating from understanding to implementation. Syntax was highlighted as an element of the programming curriculum that is afforded high levels of feedback through the programming environment. This provides an insight into why it may be perceived as slightly easier to implement syntax. It is however a minor shift so may not be considered to be of true significance. What can be considered important however is that it did not show a positive shift when all other elements of the curriculum did.

Conclusions and further research

An analysis of the survey data has provided a large number of insights into study habits and challenges faced by novice students. As indicated in the previous section it was clear that elements of program design proved to be among the most challenging aspects of introductory programming curriculum. Indeed the elements of the curriculum of a highly conceptual nature proved to be acknowledged as the most challenging, both from an understanding and implementation perspective.

A shift in acknowledged difficulty from understanding to implementation could also be seen in almost all parts of the curriculum. The only element not to experience this shift was syntax. This is an aspect of programming curriculum that provides a very high level of feedback to the students, possibly a reason why students feel a little more comfortable in working with programming syntax than their conceptual understanding of it.

The results presented are only a small part of more a thorough analysis of the data that is in progress. The most important direction for future research involving further surveys of students will focus on areas of the curriculum that contain concepts that have a high level of conceptual difficulty with the aim of clarifying exactly why students find these elements conceptually difficult. Students have commented on general topic areas only at this stage, therefore further breakdown of curriculum topics, particularly those relating to object oriented concept and design must be done to further investigate these problems.

This data provides an insight into student problems with the introductory programming curriculum. It is clear that issues relating to high concept areas and the limited feedback opportunities that they afford must be addressed. As feedback is inherently limited by programming environments and the like and the greatest opportunity for feedback comes from in-class assistance, consideration should be given to teaching methods that can provide feedback opportunities to the student both in and outside the classroom. A teaching method that can scaffold the student learning and guide them through a process such as program design may be invaluable to reducing the perceived difficulty of high-level concepts in introductory programming units.

References

Bennedsen, J. & Caspersen, M. E. (2007), Failure Rates in Introductory Programming, *ACM SIGCSE Bulletin*, Volume 39 Issue 2 (pp. 32-36). https://doi.org/10.1145/1272848.1272879

Bruce, K. B. (2005), Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list, *ACM SIGCSE Bulletin*, Volume 37 Issue 2 (pp. 111-117).

Crawford, S. & Boese, E. (2006), ActionScript: a gentle introduction to programming, *Journal of Computing Sciences in Colleges*, Volume 21 Issue 3 (pp. 156-168)

Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M. & Zander, C. (2006), Can graduating students design software systems?, *ACM SIGCSE Bulletin , Proceedings of the 37th SIGCSE technical symposium on Computer science education SIGCSE '06,* Volume 38 Issue 1 (pp. 403-407)

Giangrande, E. (2007), CS1 Programming Language Options, *Journal of Computing Sciences in Colleges*, Volume 22 Issue 3 (pp. 153-160)

Gross, P. & Powers, K. (2005), Evaluating assessments of novice programming environments, *Proceedings of the 2005 international workshop on Computing education research ICER '05* (pp. 99-110). https://doi.org/10.1145/1089786.1089796

Hadjerrouit, S. (1998), Java as First Programming Language: A Critical Evaluation, *ACM SIGCSE Bulletin*, Volume 30 Issue 2 (pp. 43-47). https://doi.org/10.1145/292422.292440

Hagan, D., Sheard, J. & Macdonald, I. (1997), Monitoring and evaluating a redesigned first year programming course, *ACM SIGCSE Bulletin , Proceedings of the 2nd conference on Integrating technology into computer science education ITiCSE '97*, Volume 29 Issue 3 (pp. 37-39)

Huet, I. Pacheco, O. R., Tavares, J. and Weir, G. (2004), New Challenges in Teaching Introductory Programming Courses: a Case Study, *34th ASEE/IEEE Frontiers in Education Conference* (pp. T2H-5 - T2H-9)

Lahtinen, E., Ala-Mutka, K. & Järvinen, H. (2005), A study of the difficulties of novice programmers, *ACM SIGCSE Bulletin , Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education ITiCSE '05*, Volume 37 Issue 3 (pp. 14-18)

Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas, K., Hanks, B., Hitchner, L., Luxton-Reilly, A., Sanders, K., Schulte, C. & Whalley, J. L. (2006), Research perspectives on the objects-early debate, *Annual Joint Conference Integrating Technology into Computer Science Education, Working group reports on ITiCSE on Innovation and technology in computer science education* (pp. 146-165). https://doi.org/10.1145/1189215.1189183

Powers, K., Gross, P., Cooper, S., McNally, M., Goldman, K. J., Proulx, V. & Carlisle, M. (2006), Tools for teaching introductory programming: what works?, *ACM SIGCSE Bulletin , Proceedings of the 37th SIGCSE technical symposium on Computer science education SIGCSE '06*, Volume 38 Issue 1 (pp. 560-561). https://doi.org/10.1145/1124706.1121514

Schulte, C. & Bennedsen, J. (2006), What do teachers teach in introductory programming?, *Proceedings of the 2006 international workshop on Computing education research ICER '06* (pp. 17-28). https://doi.org/10.1145/1151588.1151593

Winslow, L. E. (1996), Programming pedagogy - a psychological overview, *ACM SIGCSE Bulletin*, Volume 28 Issue 3 (pp. 17-25). https://doi.org/10.1145/234867.234872

Mr Matthew Butler
Berwick School of Information Technology, Monash University, PO Box 1071, Narre Warren, VIC, 3806
Ph: +61 3 9904 7163, matthew.butler@infotech.monash.edu.au
Dr Michael Morgan
Berwick School of Information Technology, Monash University, PO Box 1071, Narre Warren, VIC, 3806
Ph: +61 3 9904 7155, michael.morgan@infotech.monash.edu.au

Please cite as: Butler, M. & Morgan, M. (2007). Learning challenges faced by novice programming students studying high level and low feedback concepts. In *ICT: Providing choices for learners and learning. Proceedings ascilite Singapore 2007*. https://doi.org/10.65106/apubs.2007.2829