Examining Student Reflections from a Constructively Aligned Introductory Programming Unit

Andrew Cain  Clinton J Woodward

Faculty of Information and Communication Technologies
Swinburne University of Technology,
John Street, Hawthorn, Victoria 3122,
Email: acain@swin.edu.au  cwoodward@swin.edu.au

Abstract

Constructive alignment has been widely accepted as a strong pedagogical approach that promotes deep learning, however its application to programming units in higher education has not been widely reported. A constructively aligned introductory programming unit with portfolio assessment provides an opportunity for students to reflect on their learning. These reflections provide a rich source of information for educators looking to identify topical and pedagogical issues influencing student outcomes. In this work we applied thematic analysis to the reflective reports presented by students as part of their portfolio submission for an introductory programming unit. The analysis indicates several interesting aspects related to both topical and pedagogical issues. These results can be used to inform the development of constructively aligned programming units, and inform future research.

Keywords: Constructive Alignment, Portfolio Assessment, Reflection, Introductory Programming, Programming Issues, Thematic Analysis.

1 Introduction

Biggs’ model of Constructive Alignment (Biggs 1996), based upon constructive learning theory (constructivism) and aligned curriculum (Cohen 1987), aims to enhance student learning outcomes by focusing on what the student does. The challenges of teaching introductory programming are widely reported (Pears et al. 2007), and it has been argued that computer science and software engineering units need to transition to constructive alignment in order to better meet the requirements of the profession and improve student engagement with the field (Armarego 2009).

Recent work by Thota & Whitfield (2010) and our work in Cain & Woodward (2012) have demonstrated different methods for applying the principles of constructive alignment to the teaching of introductory programming. Thota & Whitfield (2010) presented a holistic and constructively aligned approach, aligning traditional forms of assessment and activities with cognitive and affective learning outcomes. We suggest that, in some respects, the use of traditional assessment methods can limit student engagement and constructive alignment. In Cain & Woodward (2012) we proposed an alternative approach using portfolio assessment to create a constructively aligned introductory programming unit. It is a similar approach to that originally proposed by Biggs & Tang (1997).

A key aspect of portfolio assessment is to encourage students to reflect on their learning (Biggs & Tang 1997). In the approach we suggested in (Cain & Woodward 2012), these reflections are captured in a reflective report that is required as part of each student’s portfolio. In these reflections students are required to discuss the pieces they have included in their portfolio, and how they relate to intended learning outcomes. Students are also encouraged to reflect on their learning in general.

By considering issues students are facing, educators are able to adapt teaching methods with the aim of improving student outcomes. In a survey by Pears et al. (2007), existing work on the teaching of introductory programming was considered and related to defined categories of curricula, pedagogy, language choice, and tools for teaching. Issues for students can be related to each of these areas. Although the last two categories are important, they are typically selected by teaching staff with little student input. The outcomes of research related to curricula and pedagogy are intrinsically more general in nature, and of use in a broader context.

A number of other studies have examined the nature of issues faced by novice programmers (Robins et al. 2003, Lahtinen et al. 2005). Some of the problems identified by Robins et al. (2003) included issues related to program design, algorithmic complexity, certain language features, and “fragile” novice knowledge. Lahtinen et al. (2005) conducted a survey of over 500 students, across a number of courses and institutions, asking them to rate various issues. Results indicated that issues with pointers and references, error handling, and recursion were comparatively ranked as more difficult than issues with selection structures, loops, and variables. Our work will build on these findings by examining issues identified by students undertaking a portfolio assessed introductory programming unit.

As part of an ongoing initiative to improve constructively aligned portfolio assessment for introductory programming, we wish to reflect on the learning outcomes presented in students’ portfolios. Student reflections provide an open opportunity to identify issues that are relevant from the students’ perspective. The investigation presented in this paper analyses issues identified in student reflections from a constructively aligned, portfolio assessed, introductory programming unit, and we provide some recommendations to help inform the development of units using this approach.

This paper first outlines the method used in con-
ducting the research, with details on the unit under investigation, the composition of the student cohort, and the analysis method used. Results from the data collection phase of the research are then presented. After the Results section, the Discussion presents our interpretation and analysis of the data, including recommendations to help inform the development of units delivered using this approach, and ideas for future research in this area.

2 Method

This section is divided into three parts to clearly describe the Introductory Programming Unit, the Student Cohort and Research Participation, and the Thematic Analysis of Reflections. In the Introductory Programming Unit section we provide details of the unit that was investigated as part of this research. The Student Cohort and Research Participation section details the student body undertaking this unit and how they were recruited to be part of this research. Finally the Thematic Analysis of Reflections section outlines the process followed to extract and analyse the data from the student portfolios.

2.1 Introductory Programming Unit

The unit investigated in this work was a first year, first semester, programming unit. The design, development and delivery of this unit followed the principles we outlined in Cain & Woodward (2012). This involved the definition of Intended Learning Outcomes, supporting Teaching and Learning Activities and Portfolio Assessment with Iterative Feedback. Each of these are discussed in the following sections as context for the thematic analysis.

2.1.1 Intended Learning Outcomes

The Intended Learning Outcomes (ILOs), listed in Fig. 1, were central to all aspects of the unit. They formed the central focus for students, who were required to prepare a portfolio to demonstrate they had met these outcomes by the end of the unit. The ILOs guided teaching and learning activities, which were designed to help students build skills and to give them opportunities to develop work that could be included in their portfolios.

This unit aimed to introduce students to structured programming and this is reflected in the particular wording of the ILOs.

To allow students to explore these concepts a modern version of the Pascal programming language (Wirth 1971, Van Canneyt & Kl¨ampfl 2011) was used, with a brief demonstration of the C programming language (Ritchie et al. 1978) toward the end of the unit.

2.1.2 Teaching and Learning Activities

Teaching activities took place in scheduled lectures and laboratory classes. The semester was thirteen weeks, twelve of which were teaching weeks, and a single week semester break in week eight. Topics for the twelve lectures are shown in the following list.

1. Programs, Procedure, Compiling and Syntax
2. User Input and Working with Data
3. Functions, Procedures, and Parameters
4. Branches and Loops
5. Custom Data Types
6. Functional Decomposition
7. Case Study
8. Pointers and Dynamic Memory Management
9. Structured Programming
10. Recursion and Backtracking
11. Portfolio Preparation
12. Review and Future Studies

The unit’s delivery included an early introduction topic of “understanding syntax”, where students were taught how to read programming language syntax using the visual “railroad” diagram syntax notation (Braz 1990). This allowed later lecture topics to focus on concepts, with syntax being offloaded to programming demonstrations and supplied notes, which included railroad diagrams and small code examples for each programming statement.

Allocated classes were designed with the goal of actively engaging students. Lectures typically included a review of previous topics, a short presentation using “Beyond Bullet Points” style lecture slides (Atkinson 2007), an interactive programming demonstration, and group activities. Laboratory sessions involved code reading activities, guided coding activities, and practical hands-on exercises.

2.1.3 Iterative Formative Feedback

Assessment in the unit included both formative and summative forms. Weekly assignments were submitted for formative feedback, with summative assessment of a portfolio submitted in the two week examination period that followed the thirteen week semester. A student’s final grade was determined using criterion referenced assessment (Biggs 1996).

A high importance was placed on the iterative, formative feedback aspect of the portfolio assessment process (Cain & Woodward 2012). In this unit, students were required to submit their attempts of the weekly assignments at the start of each lecture. These were collected and marked by the unit’s tutors, with the lectures being scheduled so that these exercises could then be returned to the students that same week. This enabled students to benefit from the feedback in a timely manner.

To enable the short timeframe between submission and return, tutors were instructed to focus on key issues, rather than all issues, apparent in the submitted work. This was intended to ensure that students received feedback that was relevant to them, and focused their attention on the most important areas they needed to improve.

Weekly assignments were submitted on paper, rather than electronically, to permit the tutors to rapidly review the documents and to encourage students to submit something of substance. Tutors quickly scanned through each submission looking for good qualities and issues to raise with the student. Sections of the code and answers were highlighted and then discussed directly with the student when their assignment was returned to them in the laboratory session.

No marks were allocated to these weekly assignments. They provided an opportunity for students to develop work to include in their final portfolio. This process allowed students to make mistakes without fear of losing marks from their final grade. It was expected that students would improve on their earlier
1. Read, interpret, and describe the purpose of sample code, and locate within this code errors in syntax, logic, style and/or good practice.

2. Describe the syntactical elements of the programming language used, and how these relate to programs created with this language.

3. Write small programs using the language provided that include the use of arrays, pointers, records, functions and procedures, and parameter passing with call by reference and call by value.

4. Use functional decomposition to break a problem down functionally, represent the resulting structure diagrammatically, and implement the structure in code as functions and procedures.

5. Describe the principles of structured programming and how they relate to the structure and construction of programs.

Figure 1: Intended Learning Outcomes for the unit investigated

<table>
<thead>
<tr>
<th>Pass</th>
<th>Credit</th>
<th>Distinction</th>
<th>High Distinction</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>53</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>68</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>78</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>HD</td>
<td>HD</td>
<td>HD</td>
<td></td>
</tr>
</tbody>
</table>

Portfolio includes:
- Learning Summary Report.
- Weekly assignment work.
- The three tests.

All tests have been passed. Weekly assignments have been completed satisfactorily. Learning Summary Report includes a reflection on what you have learnt, and clearly shows how all intended learning outcomes are addressed at least an adequate level.

In addition to meeting the Pass requirements the Portfolio includes:
- Bonus work from the weekly assignments.

Bonus work is sufficient so that at least one of the intended learning outcomes is addressed at a good level.

In addition to meeting the Credit requirements the Portfolio includes:
- Additional bonus work.
- A program of your own design and implementation.
- Experience Report on the above program.

Additional bonus work is sufficient so that in total at least two of the intended learning outcomes are addressed at a good level.

The program and Experience Report are sufficient so that at least one of the intended learning outcomes is addressed at an outstanding level.

The research report is sufficient so that at least one of the intended learning outcomes is addressed at an exemplary level.

Figure 2: Overview of assessment criteria provided to students in the unit outline.

<table>
<thead>
<tr>
<th>Adequate</th>
<th>Good</th>
<th>Outstanding</th>
<th>Exemplary</th>
</tr>
</thead>
</table>
| Pieces include programs that show the use of the following:
- Procedures to perform tasks.
- Functions to calculate and return values.
- Global and local variables.
- Command line arguments.
- Parameters passed by reference and by value.
- Simple statements including Assignment statements and Procedure Calls.
- Branching statements including if and Case statements.
- Looping statements including While, Until and For statements.
- Arrays.
- Custom data types including Records and Enumerations.
- Pointers, memory allocation and management.
- Code from a range of libraries.
- Codes divided into multiple units.
- Recursive functions and data structures.

Evidence also includes completed versions of most of the bonus programs from the programming assignments.

Evidence also includes a program of reasonable size and complexity that you have proposed, designed, implemented and tested.

Accompanying the program is an Experience Report that discusses topics related to this ILO. Specifically:
- How you have applied your understanding of programming structures and abstractions to the program of your creation.
- The different programming language structures used in the creation of the program.
- How these structures helped shape the design of the program.

Evidence also includes a Research Report related to programming structures such as:
- A report that discusses pointers, program memory layout, and memory management.
- A report on how function and procedure calls work below the surface.
- Other research topics as agreed with the convenor.
submissions and include these versions in their final portfolio submissions.

Students sat three tests during the semester, with additional classes being scheduled so that they did not take away from the teaching and learning time. As with the weekly assignments, these tests carried no marks. The first two tests were formative, with students needing to get near-perfect answers but being permitted to correct issues once they received their papers back. This permitted the teaching staff to gain a better understanding of student progress, and helped students practice completing programming exercises in exam conditions. The final test had to be completed satisfactorily in exam conditions for the student to be eligible to pass the unit. Students who did not complete the test satisfactorily the first time were able to resit the following week.

2.2 Student Cohort and Research Participation

This unit was undertaken by 84 students, 70 of whom submitted a portfolio for assessment. Participation in the research was voluntary, with informed consent being sought in lecture 11. All students who attended the lecture were required to fill in and sign the consent form, where they could indicate if they were willing to participate. To avoid any concerns regarding coercion, these forms were collected by a staff member not involved in the assessment of the unit, and stored until after unit results had been published. Students were made aware of these arrangements prior to giving consent.

Table 1 shows the number of portfolios made available to this research, the number that included comments related to the theme of “issues” and the distribution of grades. The grade distribution is also shown in Fig. 4, and will be discussed in Section 4.

![Figure 4: Distribution of grades for the full unit, for those students who agreed to participate in the research, and for those who commented on issues.]

2.3 Thematic Analysis of Reflections

Reflections in student portfolios provide a wealth of information. To help identify the themes and patterns in these portfolios it was decided to perform a thematic analysis using the process outlined by Braun & Clarke (2008). This process involves six phases (with some terminology adapted for clarity):

1. Familiarising yourself with the data
2. Generating initial themes
3. Searching for strong themes
4. Reviewing themes
5. Defining and naming themes
6. Producing the report

Familiarity with the data was obtained early in the process, with all of the portfolios being read as part of the unit assessment. At the end of the unit assessment notes were made in relation to the general issues that were raised in the portfolios and portfolio interviews.

Once the portfolios were made available for this research the initial themes were generated by revisiting the reflective component of each portfolio and looking for all explicit mention of issues the student faced.
Each new issue identified was matched to a theme and recorded in a spreadsheet. The spreadsheet software was used to collate the themes and record the portfolio details of where these issues had been mentioned, along with any illustrative comments using the students' own words.

In phases 3 through 5 the codes were grouped based on broader themes, and then into sub-themes. To ensure that all issues were reported in the results, the process we followed did not remove or ignore any issues raised. All issues that could not be grouped into an existing theme were collected together as a miscellaneous “other” theme. The Results section outlines the different themes identified, and how these themes relate to the comments raised by students in their reflections.

In the reporting of this analysis we present the raw coded results, grouped into the identified themes. Illustrative quotes from the student reflections are provided to help define the themes. Additional supporting evidence is also taken from the reflections of the teaching staff.

3 Results

A number of themes emerged from the analysis, and can be broadly classified as either general learning issues or programming related issues. (See Table 1 and Fig. 5.) Each of these categories is presented in Table 2 along with the number of students who raised these issues, broken down by grade. The following sections describe the individual themes in more detail.

![Student numbers for key themes](image)

Figure 5: Number of students mentioning learning issues and programming issues. See Table 1.

3.1 General Learning Issues

The general learning issues capture all of the comments made by students that do not relate directly to a given programming topic or technical aspect of the unit, but instead relate to the students’ learning experience in general. In this category the themes that appeared include time management, getting started with the unit, and learning through mistakes. The issue counts and grade distribution of these are included in Table 2, and can also be seen in Fig. 6.

Time management issues identified in the students’ reflections included comments about aspects such as “staying on task”, wishing they had “asked for help earlier”, or the general need to improve their time management to enable them to achieve higher grades. It can be seen that the majority of these concerns were raised by students who obtained either a Pass or Credit grade. These comments are further supported by observations from teaching staff, who noted concerns about students not working consistently through the semester and not seeking help in a timely manner.

The next largest general learning issue was getting started with programming. These comments specifically indicated issues related to the initial hurdle of getting started with the unit. One student noted this as their first experience using a computer, while others commented on the difficulty of the first few weeks’ lab exercises. Again, these findings are supported by observations from teaching staff who noted that a large number of students withdraw from the unit before census date, and there was a general drop in enrolment numbers around this time. This may indicate that a larger number of students faced these issues but did not continue with the unit, though further work is needed to verify this.

The last main issue in this section related to students reflecting on the mistakes or struggles that provided them with an opportunity to learn something important, referred to as learning through mistakes. For example, one student’s reflection noted that:

“...I suddenly gained insight [into the code] I had been struggling with . . .”

The reflection continued on to comment that having overcome these issues the student gained a clearer understanding of the concepts taught up to that point, and that subsequent programs were easier to understand.

A number of other issues were identified by individual students. These issues included:

- transitioning to university life and study,
- finding information in the online learning management system,
- seeking help in general,
- keeping up with the pace of the unit, noted as “challenging but good”, and
- adjusting to portfolio assessment.
Table 2: Issue count results for grade and theme. Values of interest are indicated using bold format.

<table>
<thead>
<tr>
<th>Theme</th>
<th>Description</th>
<th>Total</th>
<th>HD</th>
<th>D</th>
<th>C</th>
<th>P</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Issues</td>
<td>Issues related to learning in general</td>
<td>14</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>- Time Issues</td>
<td>Time constraints, or issues with time management</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>- Getting Started</td>
<td>Comments relating to initial weeks, or tackling early hurdles</td>
<td>7</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>- Learn through mistakes</td>
<td>Specifically commented on having issues and learning from these.</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming Issues</td>
<td>Issues related to programming topics, or technical areas.</td>
<td>11</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>- Pointers</td>
<td>Use of pointers and dynamic memory allocation functions.</td>
<td>8</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>- Parameters</td>
<td>Mentions parameters, or parameter passing</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>- Program Design</td>
<td>Algorithm and program structure design</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>- Other (Syntax)</td>
<td>Other issues, but related to the language syntax or concepts.</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>- Other (General)</td>
<td>Other programming issues not allocated to other themes.</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 6: Number of students mentioning issues related to learning. See Table 2.

3.2 Programming Issues

As already mentioned, fewer students commented on programming or technical issues in their reflections than the more general learning issues. The programming sub-themes matched specific topics covered in the unit, including pointers, parameters, program design, and recursion. In this theme the other sub-theme featured more prominently, with a larger range of issues being located in the reflections of only one or two students. The data for these themes is listed in Table 2 and shown in Fig. 7.

Amongst the identified programming issues, pointers featured most prominently. Comments typically just referred to having issues with “pointers”, with the more detailed comments discussing issues with knowing when to dereference pointers and being unsure of when to use pointers. This is further supported by notes from teaching staff indicating that pointers tended to be problematic even for students who demonstrated strong programming skills up to this point in the material.

Parameters were also mentioned by a number of students as being a topic that was particularly challenging. This included comments relating to tracing parameter values through a number of function or procedure calls, and issues of a single value having different names across different routines. From these comments there is a direct connection from parameter issues to a student’s understanding of program structure, or more importantly execution flow.

Issues relating to Program Design were also raised in the portfolio reflections. These comments related to aspects such as using functional decomposition, planning program structure, and designing algorithms.

The other issues for the programming category captures issues identified by one or two students. These were classified as relating either to syntax and concepts or general programming issues, and were:

- Syntax issues included:
  - iteration and working with loops,
  - using arrays (two comments),
  - creating composite data types using records,
  - functions in general,
  - dealing with syntax errors, and
  - using units to divide programs into multiple files.

Figure 7: Number of students mentioning issues related to programming. See Table 2.
• General programming issues included:
  – “Programming in general”,
  – “Following program code” in code reading exercises,
  – difficulties finding and using resources from the provided API, and
  – the math needed to achieve programming tasks.

There were also a number of reflections that raised the topic of recursion; these mentioned issues with both recursive functions and data structures.

4 Discussion

4.1 Investigation Focus and Sample Quality

Comments provided by students, when reflecting on their learning during any unit, can be valuable and interesting in many ways, especially with respect to the evaluation of a particular approach to teaching. Our investigation focused specifically on the theme of issues mentioned or identified by students in their reflective reports. Results of the thematic analysis, presented in Section 3, identified clear key themes. Additionally, several individual comments were selected.

The analysis considered a sample of reflective reports presented in a single semester unit. Of the 70 students in the class, almost 85% were willing to participate. Within the participant group, 35 students wrote one or more comments that matched the target theme. Table 1 and Fig. 4 show that the relative distribution of grades in the contributing group matches closely to both the participant group and the entire distribution. The consideration of this sample compares and contrasts with the results of the thematic analysis in Section 3.

4.2 General Learning Versus Programming Issues

Beginning with the two key themes of general learning issues and programming issues (Table 1 and Fig. 5) it can been seen that the distribution of student grades is very similar, with a slightly stronger representation of Pass students in the learning issues theme.

Overall, more students commented on learning in general. This is of particular interest given the relative emphasis of the course material, which focuses on teaching programming concepts over syntax details. Despite the relatively small time spent on syntax, students did not mention having related issues.

A closer examination of the issues related to programming strengthens this analysis further. Most student comments on programming issues (Table 2) concerned applying programming concepts, rather than issues of understanding syntax. Also, these comments were about when and how to use the related programming concepts rather than specifically how to apply the syntax of the language used.

Comparison of the grade distributions within the learning issues (Fig. 6) and programming issues (Fig. 7) suggests potentially interesting differences, such as issues specific to grade groups, and other issues across all grades. The sample size of this investigation limits any significant insight although some points are listed in later discussion.

4.3 Learning Issues

4.3.1 Time Management

Time management issues were identified by the largest number of students (Table 2). The grade distribution is skewed towards student’s who achieved Pass and Credit results (bold values), suggesting that students who do achieve Distinction or High Distinction results managed time better, and that the unit structure requires good time management to achieve these outcomes.

Developing a portfolio that demonstrates the ability to apply concepts taught requires time: time to practice using the concepts, and time to demonstrate their use competently. For students to achieve Distinction and High Distinction grades, they need to be able to organise their time effectively.

With more traditional forms of assessment, marks can be used as incentives. Using assessment due dates during the delivery of the unit has the effect of turning marks into time distributed weighted incentives.Marks no longer represent the importance of the learning outcome, but match allocation of incentive. Consider, for example, the allocation of marks for lab attendance. These marks do not help measure the students’ learning outcomes, but are purely there to incentivise lab attendance. Similarly, assignments due within the unit delivery periods assess the speed of acquiring the required knowledge.

With portfolio assessment the summative assessment is delayed until after unit delivery. This has the benefit of providing a more direct assessment of learning outcomes, but has a cost related to loss of incentives during delivery. While this is positive from a learning perspective, it can easily lead to students delaying their work on portfolio assessed units in order to address the more time critical assignments in other units. Given the number of comments related to this issue, it appears to be easy for students to then lose sight of how they are falling behind in a unit with a relatively flexible portfolio assessment.

4.3.2 Getting Started

Getting started is another issue facing many students (Table 2 and Fig. 6). In the first few weeks of the semester, students will face practical and conceptual issues. Practical issues include installing compilers and text editors, learning to use command line tools, and issues with general computer use. At the same time students need to build a viable conceptual model of computing (Hoe & Nguyen-Xuan 1990), and relate this to the programs they are creating.

Early on students may also face challenges transitioning to university study and university life in general. In the first few weeks students are also more likely to have issues with syntax, and dealing with syntax errors. Together these challenges can present a significant hurdle for students.

These issues could be addressed in a number of ways. Shifting toward an IDE could remove some issues related to the use of the command line compiler, but add overhead related to use of a more complex programming environment, and do not assist students in building their conceptual model of computing. The teaching staff also felt that students undertaking this unit do need to learn to use the command line, and this early introduction meant that later units could expect at least some familiarity with command line tools.
4.3.3 Learning Through Mistakes

The students’ active role in building their own conceptual model of a topic plays a significant role in constructive learning theories (Glaserfeld 1989). Effective teaching then becomes the ability to place students in situations where errors in their understanding can be challenged to help the students build viable conceptual models.

With this in mind, it is interesting to note, as shown in Table 2, the number of students who commented on gaining significant understanding through making mistakes. In line with constructive thinking, these students encountered situations in which their conceptual model was inappropriate, and in addressing the associated problems they were able to gain a better, more robust, conceptual model.

Comments about learning through mistakes were distributed across all grades, from Pass through to High Distinction (Fig. 6). This suggests that mistake-based learning experiences are beneficial to a wide range of students, albeit with some gaining a better understanding than others through the process.

4.3.4 Other Learning Issues

From the other issues students noted, many can be attributed to transitioning to university education. Learning to locate and use learning resources and to seek help, are all issues that students must come to deal with when shifting to university education.

It is interesting to note that one student did raise a complaint about portfolio assessment, indicating that it would be easier to sit an exam. While this is only a single student, it does highlight that the purpose of the ongoing assessment may not be realised by all.

Tang et al. (1999) indicated that students tend to apply narrower learning strategies for examinations, focusing on memorising material covered in lectures. In contrast, Tang et al. (1999) found that with portfolio assessment students adopted a wider perspective, making use of higher cognitive activities such as application, relation, and reflection. Students are likely to find these higher cognitive activities more challenging, and therefore those who wish to apply surface learning approaches are likely to prefer other assessment strategies.

4.4 Programming Issues

4.4.1 Pointers and Recursion

Our results support those from Lahtinen et al. (2005) in indicating that students find learning pointers challenging. Issues related to using pointers and memory management featured across a range of grade results (Table 2 and Fig. 7), indicating that this concept was challenging even for those students who managed to achieve good results in the unit.

Pointers require a good conceptual understanding of computing, and the ability to debug logical errors. Issues with pointers can often result in abrupt program termination, which can be very confronting for beginner programmers. Locating the cause of these errors is an additional challenge, that requires a student to build a mental model of what is happening within the programs they have written.

Issues with recursion were raised by fewer students than other issues, which is in contrast to the study by Lahtinen et al. (2005). This may be explained by the short time students had with recursion in this study. A deep exploration of recursion was not required for students to pass the unit. It is likely, therefore, that many students may not have had sufficient time to explore more complex applications of recursion.

In addition to being complex, pointers and recursion both occur relatively late in the curriculum. With pointers, students had little time to develop the skills necessary to handle associated issues, whereas with recursion the short time meant students had little opportunity to develop programs of sufficient complexity to encounter issues. In either case, at the time of writing their reflections, issues with later lecture topics are perhaps more likely to be in focus.

4.4.2 Parameters

Parameters require students to understand local scoping of variables, procedure and function calls, and methods for sharing these values between functions and procedures. This appears to be another point at which students need to expand their model of computing (Hoc & Nguyen-Xuan 1990).

While parameter concepts can take time to understand, issues are likely to be constructive in nature. When the logic for a program is contained within a single procedure, students can develop a simplistic model of what is occurring when other functions or procedures are called. When students need to design their own functions and procedures that require parameters, they are presented with situations that challenge their simplistic model. This suggests that parameters provide a significant learning opportunity from a constructive perspective.

The two different parameter passing methods are both taught in the unit, with pass by reference being used to create procedures to swap parameter values, as well as allowing procedures to modify data within structures and arrays. Call by reference provides an early introduction to references.

4.4.3 Program and Algorithm Design

Program and algorithm design are progressively taught throughout the unit, with the main focus being in the middle of the unit’s delivery in topics related to functional decomposition and structured programming. Comments by students indicated several issues on how to practically apply the concepts covered to create programs.

The authors of this paper initially expected a larger representation of this issue, as design tasks require a deeper, relational, understanding of the concepts being used. However, the core tasks students had to submit for a Pass grade were accompanied with detailed instructions to help ease these design issues. Extension tasks required for a Credit grade did require some design components, and less guidance was provided. Students attempting their own program, necessary for a Distinction grade, needed to perform design activities as these programs were of their own design and creation.

4.4.4 Other Programming Issues

The other programming issues raised by students can be classified as individual challenges. It seems that students are likely to learn at different paces, in different ways, and find different topics challenging. Again, general comments concerned the application of programming concepts, rather than with basic syntax.

Each of the raised issues indicated a point at which students had an opportunity to challenge and develop their conceptual understanding of programming and their model of computing.
4.5 Recommendations

Based on the thematic results and on the experiences of staff involved in the unit delivery, there are a number of implications and recommendations that can be made. These recommendations are listed below, and will be explained in later sections:

- strongly avoid mixing formative with summative assessment,
- give students time to adjust to portfolio assessment,
- focus on student “awareness”,
- use a quick formative feedback process,
- avoid the “tutor debugging” phenomena,
- use visual methods to convey progress, and
- make students aware of issues they are likely to face.

4.5.1 Always formative, lastly summative

Separating formative feedback processes from summative marking has a clear value, and this is reflected in student comments. Our observation is that using a punitive marking system creates an incentive for students to hide faults and limits in their understanding. Students need to know what they need to learn. Related to this is the time a student can spend asking about marking schemes or lost marks – time better spent on learning.

4.5.2 Students need time to adjust

In comments to staff, students have said that it takes time to get used to a portfolio based unit even if they understand the principles. If we consider that students might be conditioned to respond to summative marking and due dates as a way of allocating their attention, an interesting question emerges: how do we help students maintain an active engagement with the unit activities? Finding an answer for this is an ongoing challenge and research opportunity.

4.5.3 Focus on student awareness

Primarily, student awareness is the basis for positive engagement and an aware student has the opportunity to make appropriate choices. To support this, staff need to communicate the structure, activities and expectations of a portfolio-based unit to students as effectively as possible. Unfortunately students may essentially have habits that can take time to adjust. It is possible to help students with issues such as time management and, hence, learning outcomes.

Although formative activities may not have due date or marks (grade penalties), staff should still express clear expectations of when work needs to be done. In some cases this leverages a students’ habits to their advantage as they feel compelled to do the work. Ideally, students should give these formative tasks as high a priority as assignments with marks.

4.5.4 Use quick formative feedback

Very quick feedback helps to create strong reinforcement in a student that the process really is formative and personally valuable. In a students’ experience summative marking is often a delayed process. If formative feedback takes a long time it is removed from the students’ current learning and challenges, and so can be confused as summative marking. Students need to be engaged with the formative nature of these assessments, making use of the feedback to help develop their understanding.

4.5.5 Avoid tutor debugging

A possible problem with quick formative feedback, and resubmission opportunities, is that students may submit poorly prepared “drafts” and use staff simply to “fix things”. This issue has been described as “tutor debugging” by some of our staff, and should be actively discouraged. One approach to this is to set minimum submission standards for work submitted for feedback.

4.5.6 Use visual methods to convey progress

Visual charting of tasks and completed work, calendar events and strong reminders of work and time limitations help to engage students. It is also possible that a “gamefication” approach, by recognising personal or group achievements and rewarding with awards, “badges” an other game-related concepts, can create a fun and personal incentive for students. We also recognise that there are also risks with gamefication, such as trivialisation of the value of core learning activities or distorting the value of learning activities through association to a gamefication artefact.

4.5.7 Tell students what to expect

Finally, helping students understand the issues they are likely to face should help them prepare sufficiently for the more challenging tasks. This is particularly relevant to the issues related to getting started. The challenges early on in the unit may put a number of students off, and these students are likely to lose motivation and engagement with the unit. Making them aware that these challenges are “normal”, and to be expected, may assist them in getting over early hurdles.

4.6 Future Work

It would be valuable to compare the thematic results between different semester groups undertaking the same unit of study to see if the results are similar. We would also like to compare results from this programming unit to later programming units. Similarly, as programming units are undertaken by students from different courses, each with different expected aims and outcomes, it would be interesting to see if the themes identified correlated to particular groups. Ideally this could inform both the development and delivery of programming units, as well as courses as a whole.

Once students are familiar with the learning environment, and expectations of a unit delivered using constructive alignment and portfolio assessment, it would be reasonable to expect some indication of this in the reflection comments presented by students. Future investigations could look for changes in themes such as getting started or time management which would ideally improve in students with experience.

While this investigation focused on the issues identified by students, there are other themes that could be used as a focus for thematic analysis. Themes could be compared to the learning modes or preferences of students, which we would expect to strongly correlate to reflection themes in some cases.
5 Conclusion

In this paper we have presented a thematic analysis of reflective reports presented by students as part of their assessment in an introductory programming unit. The development and delivery of the unit was described in detail as a context for the work. A good representation of students distributed across all result grades agreed to participate in the study. Thematic analysis was directed specifically at the theme of issues identified by students. Overall results showed that more students raised learning issues than programming related issues. Significant learning themes included time management, getting started and mistake-based learning. The most common programming issues were related to pointers and parameters, with only a small number of issues related to syntax, and both these results were expected. Issues related to program design were raised less than expected.

The discussion considered a number of interesting results, and put forward recommendations and future directions for research in this area.

Notes

1Unit in this context refers to a course/subject/module within a degree programme.
2This research was granted ethics approval in accordance with [institution details anonymised].
3Structured programming in the context of the ILOs has a broad meaning, encompassing imperative, procedural, and structured design concepts, as well as control flow.
4This is the date when the university records enrolment numbers, typically a few weeks after the start of the semester to allow for changes of enrolment.
5Students were provided with an API for creating small games. This included functionality for drawing shapes and images, playing sound effects and music, handling input, and other functions and procedures related to creating small 2D games.

References


Atkinson, C. (2007), Beyond bullets points: using microsoft@office powerpoint®2007 to create presentations that inform, motivate, and inspire, Microsoft Press.


URL: http://dx.doi.org/10.1007/BF00869951


