The PTP Model: a predictor for computer programming success

Oliver Kerr and Nicky Danino
University of Central Lancashire, Preston, Lancashire, United Kingdom
OAPKerr@uclan.ac.uk *NDanino@uclan.ac.uk*

Abstract

The fundamental concepts of programming are essential to any Computer Science course yet, these concepts can appear significantly more abstract than students have encountered in the past. These abstract concepts can become so daunting to students, that they experience 'programming shock' during their first encounter with programming, as they attempt to decipher a number of concepts, error messages and unfamiliar syntax all at once. Once a student encounters programming shock, it can be extremely disheartening and if not overcome, can sometimes result in a student dropping out from a course. Through the use of specifically designed aptitude tests conducted with first year Computing students, this investigation has provided sufficient evidence to prove a link between mental model usage and student performance in an introductory programming module, as well as enabling the development of the Programming Thought Process (PTP) model, which can be used to identify students most in need of support.

1. Introduction

Essential to any Computer Science course are the fundamental concepts of programming, yet, despite the importance and popularity of Computer Science as a subject, it suffers some of the highest dropout rates of any university course [7]. Research shows that issues with retention in Computer Science courses could potentially be due to the difficulties that students face when learning to program [17].

The aim for this research is to build on previous research [9] and to improve understanding of the programming learning process. This involves developing the ability to deduce potential predictors of success or failure within introductory programming modules, which in turn allows for students who are most likely to struggle with programming concepts to be identified, and given the support they need to succeed in their degree.

2. Background and related work

2.1. Computer Science in education

Before expanding on the theoretical foundation of this study, it makes sense to understand where Computer Science is situated in education learning.

Despite the prevalence of IT and technology in our modern society, the UK is currently suffering a significant digital skills gap due to a lack of interest in ICT [3]. In an introduction to the Royal Society report entitled, "Shut Down or Restart? The way Forward for Computing in UK Schools" Professor Steve Furber [6] notes that Computer Science has been included in the National Curriculum under the heading 'ICT – Information and Communications Technology' since the turn of the century, which was introduced by the government in order to improve digital literacy in order to meet business needs. ICT has remained largely the same since its introduction, covering topics aimed at providing students with basic computer skills. However, current generations of students; who can be considered 'Digital Natives', have grown up surrounded by computers, video games, smartphones etc. and as such, are constantly exposed to technology. This constant exposure to technology has begun to change the way students think and process information [16] and has led to a common opinion amongst students that 'ICT is boring' due to the lack of challenges and stimulation is provide [3]

The lack of interest in ICT by students is contributing to a digital skills gap that is costing the UK an estimated £63 billion a year in lost GDP [3]. In order to address the skills gap, the government introduced a new Computing curriculum as well as a Computer Science GCSE in 2014 [4].

It is generally accepted that an outcome of any Computer Science course is that a student develops the ability to program. However, the stark differences between Computer Science and ICT has resulted in a shortage of specialist teachers who are capable of teaching the subject, with only a third of ICT teachings holding relevant qualifications and only 25% feeling confident enough to teach the curriculum [3].

Whilst learning to program, students encounter a variety of abstract concepts [17] that may require significant explanation from a teacher in order to effectively communicate them to a student. However, if the teacher themselves is struggling to understand

these concepts, then it is unlikely that they will be able to communicate them correctly to students.

The issues relating to the programming learning process are discussed in the subsequent sections.

2.2 Psychology of programming

In order to fully appreciate and understand the programming learning process it is important to explore the psychological factors that can affect a student as they attempt to learn to program.

Lahtinen et al. [10] identify the act of learning to program as one of the biggest challenges of studying Computer Science. Du Boulay [5] provides an overview of overlapping domains and potential difficulties encountered when a student is learning to program. They are as follows:

1. General Orientation – what programs are used for and what can be done with them.
2. Notional Machine – a model of the computer as it relates to executing programs.
3. Notation – the syntax and semantics of a specific programming language.
4. Structures – applications and adaptation of known schemas and plans to suit the requirements of the program.
5. Pragmatics – the skills of planning, developing, testing, debugging, etc.

Du Boulay [5] goes on to determine that none of these issues can be fully separated from the others, resulting in students experiencing 'shock' during their first encounters with programming as they attempt to try to deal with all the issues at once.

Rogalski and Samurçay [17] also acknowledge that learning to program is an extremely complex process, involving a variety of cognitive processes, and mental representations in order to develop conceptual knowledge and structuring of basic operations (such as loops, conditional statements, etc.) into schemas and plans. Winslow [20] states that it takes approximately ten years to turn a novice programmer into an expert, meaning that a three-year undergraduate course can only provide the foundations that a student needs to develop into an expert on their own. By introducing the fundamentals of programming at a much earlier age; as was suggested by Blackwood [3], the majority of students would already have a basic understanding of programming – allowing for more detailed study at university level and employability.

In order for a programmer to understand what the code is doing they must develop a '*Mental Model',* based on their '*Domain Knowledge'* [1] – their knowledge of related topics; i.e. Mathematics, Physics or previous programming experience. Experienced programmers will have encountered a wide range of scenarios before, allowing them to develop reliable mental models. However, a novice programmer does not have the same amount of experiences to draw on; meaning they must rely on adapted knowledge from other subjects, often resulting in inconsistent and incorrect mental models being produced [20].

Johnson-Laird is considered to be one of the pioneers of the Mental Model theory, which is viewed as one of the most influential theories in cognitive psychology [18]. Johnson-Laird [8] states that many humans claim to have the ability to form mental representations in the absence of corresponding visual stimuli. It is unlikely that these are simple pictures or descriptions that can be defined by true or false. Johnson-Laird [8] claims that humans see the world through established mental models and develop an understanding of a proposition that is true or false to a respective model.

At the beginning of a programming course students will likely have a number of pre-established mental models. Rogalski and Samurcay [17] note that when a student begins to program, they commonly refer to everyday objects to allow them to visualize what the program is doing and that students who have studied subjects such as Mathematics or Physics often find it easier to form mental representations, and to develop an understanding of abstract concepts. It is likely that students have already developed appropriate mental models from the related subjects, which are then applied to the programming scenario.

Mental Models are crucial to building understanding, if a teacher neglects to omit them students will make up their own models of dubious quality [20].

Norman [15] goes on to identify a number of characteristics of mental models:

1. Mental models are incomplete and simplified, due to limited knowledge and expertise a complete model can be difficult to construct.
2. People's abilities to 'run' their models are severely limited.
3. Mental models are unstable.
4. Mental models do not have firm boundaries; similar operations can become confused.
5. Mental models are 'unscientific', people follow their set behavior patterns.

These characteristics highlight how mental models are often unique to individuals due to their past experiences (domain knowledge) and how their models can be adapted over time as more experience is gained.

In the context of this research investigation, mental models are used to represent how students approach variable swapping questions as part of a programming aptitude test.

3. Research design

This research takes a mixed method (methodological triangulation) approach and uses multiple data collection techniques. This allows for the 'true results' to be established through triangulation.

Purposive sampling is used to collect data for this investigation, as it allows a specific group of participants to be selected to take part in the research, which in the context of this research experiment, is first year Computer Science students studying in either Preston or Cyprus. By including all first year Computer Science students in the investigation, it allows for an exploration of the effects of domain knowledge on mental model usage, as the uniqueness of every student and their experiences makes establishing a representative sample group difficult.

First year Computer Science students at the University of Central Lancashire (UCLan) come from all manner of academic backgrounds. Many have studied either Computer Science or IT at school or college, whilst others have not undertaken any formal qualifications in the subject. Computer Science demands entry requirements of up to 280 UCAS tariff points at A2 or BTEC National Diploma MMM-DMM AND 5 GCSEs at grade C or above including Maths and English. However, there are not specific requirements to have previously studied a computing-based subject. Students study the first year to gain a grounding foundation in Computer Science; including an introductory programming course, and then progress to year 2 where they choose a specialism. Specialisms range from Computer Games Development, Computer Network Technology, Information Systems, Forensic Computing, Software Engineering, and Computer Science, which is a student self-select course that offers a flexible programme of study.

The entire premise of this research study is based around understanding the programming learning process. Naturally, this draws on the authors' epistemological standpoint as students who have completed a number of programming and Computer Science related courses in the past. We believe that, like the theory of Constructivism [2], programming ability is obtained through practice and cannot be effectively transferred through a traditional lecture. Whilst a lecture is a useful medium for teaching complex concepts such a polymorphism, when it comes to the fundamental concepts of programming, the best way to develop a concrete understanding is for students to spend less time getting confused in lectures and spend more time putting the concepts into practice.

4. Method

4.1. Survey

A preliminary survey was carried out to establish the domain knowledge of the students taking part by developing an understanding of their past experiences. The survey was made available online and begins by asking students to list any previous courses they have studied (post GCSE level). This allows for links between mental model usage and subjects to be established, for example, a student who has studied Physics may have a different model to someone who studied English literature, hence developing a different domain knowledge that affects how they approach a given scenario.

To allow for easy analysis students were asked if they have studied Computer Science at any level, (can only answer yes or no) as the sample size is so large it would be easy to miss a subject if they were being categorised manually. The final section asks students if they have had any prior programming experience and if so to list the languages they are familiar with. This is an extremely important question as past programming experience could impact on mental model usage and some students may have studied programming in their own time, which would not have been highlighted in the first question. By conducting the survey online, it allows participant's data to be easily anonymised as well as allowing for easier analysis, as all the data can be downloaded and imported into an Excel document, rather than inputting it manually, reducing the risk for human error in the data.

4.2. Aptitude test

The aptitude test forms the main focus of the data collection process. The test is an online application and was delivered within timetabled classes. Students were informed about the purpose of the research, what activities they would be taking part in, as well as what data would be collected and how it would be used. It was made clear to all participants that they could opt out and remove their data at any time during the experiment. They were also given the opportunity to ask any questions and discuss the investigation at the end of the sessions, and they also had the opportunity to check any of the data they had provided.

The aptitude tests were conducted twice, once at the start of the main teaching block, (week 6 in the first semester due to introductory activities) and once in the final week before Christmas (week 14). The preliminary survey; which collects information regarding the students' background and past experiences (domain knowledge), was conducted at the same time as the first aptitude test. All tests were carried out in controlled conditions during timetabled sessions. A pilot study was also conducted with five

students in order to fine-tune the questions and the aptitude test application.

Students participating in the study are all at the same level in their course, testing at these two intervals allows us to track their development throughout the first semester – which for many of them, will be their first semester programming.

As well as mental model usage, the variable swapping questions in the aptitude test allowed us to investigate two other factors – juxtapositions, and the consistency of use. Several of variable swapping questions involved multiple variable assignments and re-assignments, in order to produce a correct answer, students must understand the concept that a variable is held within computer memory and can be modified.

The consistency of a student's mental model and juxtaposition usage is an important factor in this research, as it allows for a student's understanding of programming concepts to be evaluated. As Johnson-Laird [8] states, there can be multiple valid mental models for a given scenario however, only a single model would be appropriate in the context of programming as the compiler will only execute a statement in one particular way, regardless of how a student interoperates it. For example, even a simple assignment operation such as 'A = B' would have multiple valid mental models.

If the way the compiler processes a particular statement does match with a student's mental model the result will be logical errors within the code. It is therefore important to examine the consistency of mental model and juxtaposition usage in order to identify students who have successful understood a particular concept, those who are beginning to understand it but still require support and those who are struggling to get to grips with a particular concept.

A full breakdown of the identifiers used to represent the mental model, juxtaposition and consistency levels can be found in Table 1. Combinations of the three factors will be referred to using the identifiers listed in Table 1, for example, *m2s1c0* refers to the use of the *m2* model (the model which appropriate in a programming context), *s1* juxtaposition (variable changes are carried through to subsequent statements – appropriate for programming) and *c0* (use of a single mental model and juxtaposition (where appropriate) for at least 80% of the questions).

The naming convention for the identifiers has been adapted from a similar experiment by Dehnadi [4] to allow for easier comparison of results.

Table 1. Identifiers used to analyse students' aptitude test results

Mental Models	
Identifier	**Description**
m1	Value extracted from right to left, right value becomes 0. (a←b; b←0)
m2	Value copied from right to left. (a←b; b unchanged)
m3	Value extracted from left to right. (a→b; a←0)
m4	Value copied from left to right. (a→b; a unchanged)
m5	Right-hand value added to left. (a←a+b; b unchanged)
m6	Right-hand value extracted and added to left. (a←a+b; b←0)
m7	Left-hand value added to right. (a+b→b; a←0)
m8	Left-hand value extracted and added to right. (a+b→b; a←0)
m9	Nothing happens. (a,b unchanged)
m10	A test of equality.
m11	Variables swap values.
NA	No valid mental model used – Erroneous answer / Blank answer
Juxtaposition	
S1 - Sequence	The first assignment operation is implemented with the initial value, the second operation uses the modified value – appropriate for programming.
S3 – Simultaneous	Each assignment operation uses the initial values of the variables.
NA	No valid juxtaposition used - Erroneous Answer / Blank answer

Consistency Level	
C0	Used a single mental model for at least 80% of the questions.
C1	Used a combination of two related mental models i.e. m1 and m2.
C2	Used a combination of up to four related models.
C3	Used a combination of up to eight related mental models or any of the remaining unrelated models.

4.3. Module grades

To aid in the development of predictors of success or failure of students studying introductory programming courses, student performance must be examined. All students who took part in the investigation were enrolled on the 'Introduction to Programming' module, which aims to teach students the fundamentals of programming in the C# language. The module is assessed through a piece of compulsory coursework and an examination, each of which is weighted at 50%.

For their assignments, students were tasked with creating a simple file management program which allowed students to display files in a given location, as well as being able to filter files and output folder statistics.

The examination covered programming fundamentals, including questions on data types, recursion, iteration, if-statements and functions. A mixture of multiple choice and open-ended questions where students were asked to write code to perform a simple operation were used in the exam.

4.4. Contrasting data collection

In order to develop a detailed understanding of how domain knowledge can affect a student's mental model usage it is important that this investigation also looks outside the Computer Science subject area to help identify influencing factors.

Students who come from mathematical background (subjects including Mathematics, Engineering, Physics, etc.) often find it easier to understand the abstract concepts of programming [17] therefore, we felt it appropriate to examine the characteristics of a group of students studying a completely unrelated course – in this case a focus group consisting of students studying Business Management related courses.

The focus group was made up of 28 students in total, 23 Business Studies students (16 foundation year, 7 first year) and 5 Accounting students (4 foundation year, 1 first year). The students were given the same variable swapping questions as the Computer Science students during their first aptitude test, by doing so it presents the opportunity to examine how differences in domain knowledge affects mental model utilisation.

5. Results and discussion

5.1. Student participation

All first year Computer Science students were given the option to take part in this study, 123 students completed both the survey and the first aptitude test. 73 students stated they have previous programming experience, however, only 57 students stated that they had been formally taught Computer Science in the past, creating additional variability into the results, as Ben-Ari [2] noted that self-taught programmers do not necessarily succeed in Computer Science studies as they may have constructed non-viable models by misunderstanding concepts.

Only 67 of the original 123 students took part in the second aptitude test, this is likely due to the test being conducted in the final week before the Christmas holidays, as attendance to the classes was noticeably low. Despite this reduction in sample size there are still enough students to perform a reliable investigation with and therefore, these students will form the primary focus of the data analysis. Any trends that are found in students who completed both tests can in fact be supported by analysing the original group of students.

5.2. First aptitude test

The first aptitude test was carried out during the first week of the main teaching block, allowing us to establish the characteristics of students' mental model usage before any significant amount of teaching has taken place.

By analysing the answers students provided to the variable swapping questions, we determined that 79% of students used *m2* as their dominant (most frequently used) model. 53 students in total used the *m2* model however, 25 students used *s3* as their dominant juxtaposition. The use of the *s3* juxtaposition is significant as it means students were referring to the initial values of variables when attempting to answer questions with multiple assignment operations, instead of carrying the changes from one statement to another. This suggests to us that understanding the way variables work within a computer could be a potential threshold concept for students. A threshold concept is defined by Meyer and Land [14] as a concept that are necessary in order to progress in a subject and often transform the way a student looks at a subject however, they are also the topics where students are can get stuck.

Figure 1 displays the distribution of mental model and juxtaposition usage from the first aptitude test.

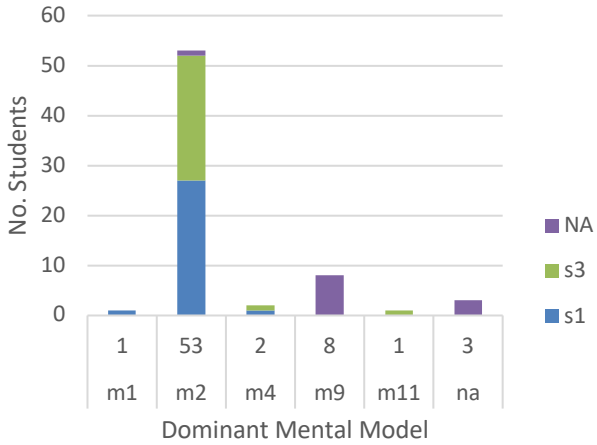

Figure 1. Mental Model / Juxtaposition usage of students during the first aptitude test

Overall students were relatively consistent in their mental model usage, with 73% of students consistently using a single model (*c0*). 40% of students used an appropriate mental model and juxtaposition consistently (*m2s1c0*) during the first aptitude test.

The results discussed above are representative of the entire sample group who participated in both tests, meaning that includes both students who have previous programming experience and those who do not. It is therefore important to further analyse the data by separating the two groups of students in order to understand how their mental model usage differs, as Soloway [19] suggested, a student's mental model is based off their past experiences, potentially making the two groups drastically different.

Of the 67 students to complete both tests, 36 stated they had prior programming experience. The most common dominant mental model was *m2* which was used by 34 students, suggesting that their pre-established domain knowledge is potentially influencing the mental model usage.

11 students who used the *m2* model also used the *s3* juxtaposition, with an additional student being classed as using an invalid juxtaposition (*NA*). This suggests that despite previous experience, understanding how variables are handled within a computer is still a potential threshold concept.

74% of students with previous programming experience consistently used a single mental model (*c0*), with all 22 students who used the *m2s1* combination doing so consistently. With 26% of students being inconsistent in their mental model usage it would suggest that even though they have prior programming experience, some students are still adapting to using an appropriate model.

When analysing the results of the 31 students who stated that they did not have any prior programming experience it was discovered that the students were relatively consistent with their mental model usage with 71% consistently using a single (not necessarily correct) model however, only 5 consistently used *m2s1*. There was also a greater variety of dominant models used by students with no prior programming experience as opposed to those with experience, with 5 different models as opposed to 3 being used respectively.

5.3. Second aptitude test

By conducting a second aptitude test at the end of the first semester, it allows the development of students' mental model usage to be investigated. We expected that students who have already adopted their dominant combination would continue to use it with increased consistency. We predicted that students who previously used other mental model combinations will begin to gravitate towards a new model because their domain knowledge begins to adapt as they progress through the course. Constructivism theory positions that students accrue their knowledge recursively by taking an active part in the learning process [2], i.e. completing practical lab sessions, aimed at teaching students the fundamental concepts of programming by putting them into practice.

This could potentially mean a decrease in overall consistency as students may experience cognitive conflict; when a student experiences a discrepancy between their own cognitive structure and an external environment [11], i.e. conflict between a student's original incompatible mental model and the appropriate model used in programming which they are attempting to learn.

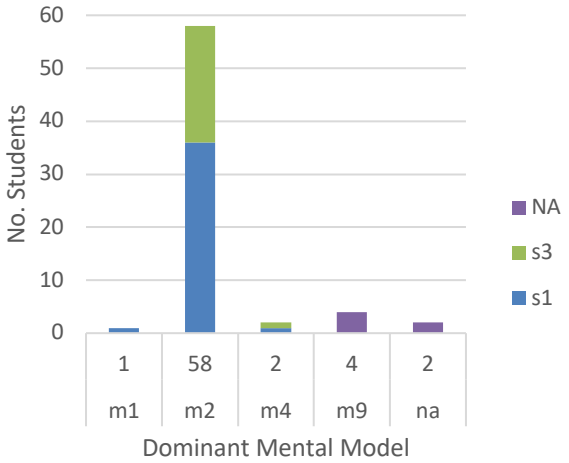

Figure 2. Mental Model / Juxtaposition usage of students during the second aptitude test

In total, 21 students improved their model usage between the two tests, either by switching from an

inappropriate model to *m2* (any juxtaposition), or by switching from an invalid model (*NA*) to a known valid (although potentially inappropriate) model. 49 students remained consistent in their usage between the two tests and 6 students became less consistent but continued to use the same dominant model, perhaps due to them beginning to adapt to a more appropriate model.

As Figure 2 shows, the variety of models used by all students in the test group has decreased when compared to the results of the first test (Figure 1). There has also been a clear uptake in the number of students using the *m2* model. In total, 58 students (86%) are now using *m2*, with over half (56%) the sample group using *m2s1*, a rise of 14% from the first test. The number of students consistently using a single model has slightly decreased from 73% in the first test to 70% in the second.

48% of students consistently used *m2s1* whilst the number of students using *m2s3* (of any consistency) has dropped slightly to 33%, supporting the idea that understanding variables is a potential threshold concept due to the large portion of students not applying an appropriate juxtaposition to the variable swapping operations.

Interestingly, there has been very little variation in the mental model/juxtaposition usage of students with past programming experience between the two tests. The number of students using the *m2* model has remained constant, however, four students who previously used the *s3* juxtaposition have now begun to use *s1*. Consistent use of a single model has also increased amongst students with previous experience from 74% to 77%. The number of students consistently using an appropriate mental model/juxtaposition combination (*m2s1c0*) has also increased from 22 to 24. This adds support for out theory that as the domain knowledge develops, a student will begin to gravitate towards *m2s1c0* due to the knowledge they have acquired, which in turn reduces cognitive conflict.

Whilst the variations between the two tests were relatively small for students with past programming experience, there are a number of significant changes amongst those with no previous experience.

The most notable difference is the variety of dominant models used by students has decreased with only 4 out of 28 students having a dominant model other than *m2*. However, only 32% of students are using *m2s1* compared to 69% of students with programming experience. The number of students consistently using a single model fell 10% between the two tests, which we believe to be a sign of cognitive conflict. Maier [12] states that before a student can fully understand a concept, a student's understanding must be challenged before they can adapt. Many of the students who had not previously had any programming experience held incompatible models, which must be first actively challenged by allowing a student to gain programming experience, thus allowing a student to discover for themselves what works, and what doesn't, in a programming context.

We believe the reduction in consistency is attributed to students' mental models being challenged, whilst in some cases they may still be using their previously established models, students should be beginning to gravitate towards the appropriate mental model/juxtaposition combination.

5.4. Module grade comparison

The final key element of this investigation is the comparison of students' mental model/juxtaposition usage and their overall grades for the Introduction to Programming module, thus allowing for predictors of success to be established. While grades are available for all students enrolled on the module, the results discussed below refer to the group of students who completed both aptitude tests. Although this limits the sample size significantly, it allows for a better understanding of a student's development through the module, i.e. a student may have originally started using an incompatible model such as *m9* but by the end of the semester they may have begun to use *m2*. For this reason, the discussed results focus on the dominant models identified during the second test.

55% of students who completed both tests obtained a 1st (70% or above), 73% of which used *m2s1c0* as their dominant combination. 9 students achieved a 2.i (60% - 69%) overall, 33% of which used *m2s1c0* and 33% used *m2s3c0*. 12 students a 2.ii (50% - 59%) in the module with *m2s1c0* and *m2s3c0* were only used by a single person each, 33% used *m2s3incon* (either *c1,c2 or c3*) whilst 25% used an inappropriate model inconsistently (*other incon*). Only 6 students obtained a 3rd (40% -49%), primarily students who achieved this grade used *m2s3incon* (33%) or an inappropriate model inconsistently (*other incon* – 33%).

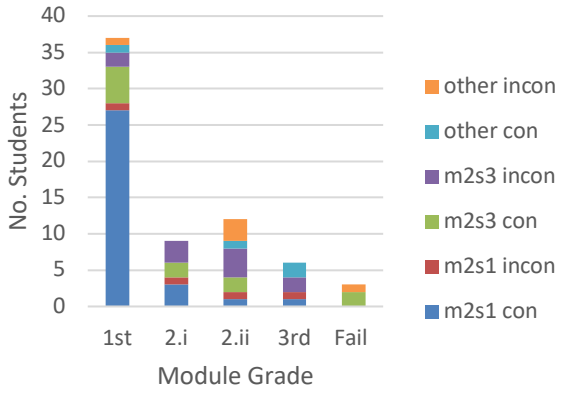

Figure 3. Comparison Mental Model/ Juxtaposition / Consistency and module grades

Three students also failed the module, two of whom used *m2s3c0* and one used an inappropriate model inconsistently. Figure 3 highlights the relationship between mental model/juxtaposition usage, consistency and module grades.

One trend that is immediately identifiable from reviewing Figure 3 is the relationship between the use of *m2s1c0* (*m2s1 con*) and students achieving higher grades. Also, as grades decreased, so did the use of *m2s1c0*. This suggests that the use of *m2s1c0* could potentially be used as predictor of success within the course

A Mann Whitney U test [13] was also carried out to evaluate the significance of the relationship between appropriate mental models and module result, producing a result of $p < 0.05$ and confirming the significance of the data.

5.5. Original participants

Whilst the results discussed previously provide an insight into the development of students throughout the first semester, the sample size is extremely limited due to not all students completing the second test. 123 students took part in the first aptitude test meaning there is a lot of reliable data available, which can be used to support the findings of the primary investigation.

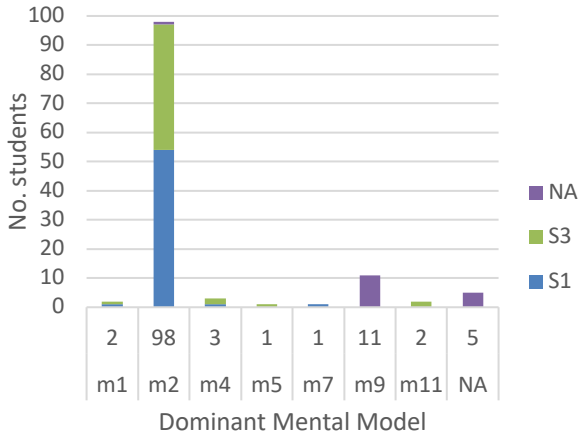

Figure 4. Mental Model / Juxtaposition usage of all students who completed the first aptitude test

Figure 4 shows the distribution of the dominant mental model/juxtaposition combinations for all students who took part in Test 1. The range of combinations used by students is significantly more varied than that observed in Figure 1, however, a commonality between the two is that *m2* is by far the most commonly used mental model as 80% of all students utilized *m2* as their dominant model. There is also an almost equal split between the number of students using the *s1* and *s3* juxtapositions, similar to that observed in the test group. Despite the varied combination uses, 68% of students consistently used a single model with 42% using *m2s1c0*. Of the 123 students who originally took part in the first aptitude test,

73 of the 123 students who took part in the first test stated that they had previous programming experience. 88% of students used *m2* as their dominant model of which 58% used the *s1* juxtaposition. Other than a relatively small percentage of students using other miscellaneous inappropriate models, students who did not use *m2s1* used *m2s3* instead (29%). The high percentage of students using the *s3* juxtaposition; despite their previous programming experience, leads us to believe that the assumption that understanding how variables are handled within a computer is a potential threshold concept for students. 70% of student who had previous programming experience consistently used a single mental model, with over half of students using *m2s1c0*.

50 students who stated that they had no previous programming experience and like the primary test group used a large variety of mental model/juxtaposition combinations. *m2s3* was the most common dominant combination, being used by 44% of student. 68% of students consistently used a single mental model/juxtaposition, however, less than a quarter used *m2s1c0*.

As only 54% of students completed both tests it would not be appropriate to use the results from the second test when comparing the mental model and juxtaposition usage to module grades for the group as a whole. However, by comparing all 123 students' mental model usage from the first test to their module grades, it allows for trends between students' initial mental model usage at the beginning of the course, and the grades they achieve to be established. This not only supports trends discovered with the test group, but also forms a solid foundation for a future implementation of the testing method, which would likely be conducted at the beginning of a course.

54 students (44%) obtained a 1st overall in the module, 18 (15%) achieved a 2.i, 18 (15%) achieved a 2.ii, 16 (19%) achieved a 3rd and 8 (7%) students failed. 8 students also had no corresponding grades, potentially due to incorrect ID numbers being entered at the start of the test.

Out of the 73 students with programming experience, 50% obtained a 1st in the module, the majority of which used *m2s1*, it was also determined that 75% of students using *m2s1* used it consistently (*m2s1c0*). A Mann-Whitney U test [13] produced a significance of $p < 0.05$, further supporting claims of a link between appropriate Mental Model usage and success in the module.

The remaining 50 students who took part in the first test and did not state that they had any pervious programming experience and as a result, their lack of experience has translated into a much greater variety of mental models being used, as well as a smaller

proportion of students achieving higher grades than students who had programmed in the past.

Only 34% of students who had no previous programming experience obtained a 1st in the module. However, this was the most commonly achieved grade with 22% achieving a 2.i, 14% achieving a 2.ii, 14% achieving a 3rd and 6% failing the module.

Despite the more varied results; when compared to students with previous programming experience, *m2s1c0* was still the most prevalent model combination amongst students achieving higher grades, with it being used by 47% of students achieving a 1st in the module.

The number of student using *m2s1c0* falls drastically as grades drop, with it being used by 27% of students who obtained a 2.i and then not at all by students achieving subsequent grades. The most prominent model combination used by students who achieved a 2.i and 2.ii is *m2s3c0*; 55% and 57% respectively, whereas students who achieved a 3rd or failed the module primarily used an inappropriate model inconsistently (*other incon)* or *m2s3c0*. A Mann-Whitney U test [13] confirmed the significance in the relationship between module grades and Mental Model/Juxtaposition combination usage by students with a significance of $p < 0.05$.

5.6. Business students

To further investigate the impact domain knowledge has on mental model usage, we ran the test with a group made up of 23 Business Studies students (16 foundation year, 7 first year) and 5 Accounting students (4 foundation year and 1 first year). Whilst both these subjects involve mathematics, we believed them to sufficiently different from Computer Science and other subjects that require an understanding of abstract concepts, i.e. Physics [17], to provide a basis for comparisons of students' mental model usage.

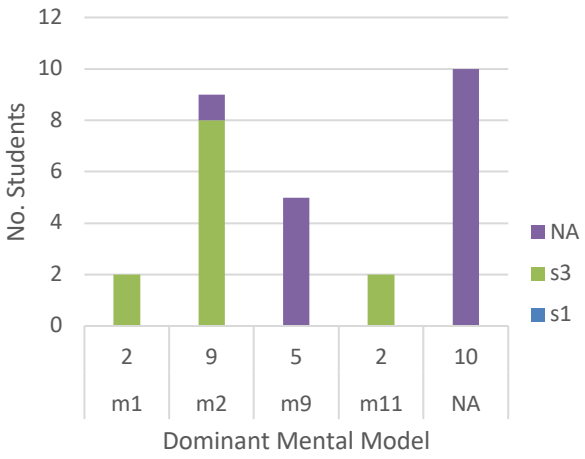

Figure 5. Mental Model / Juxtaposition usage of business students

A relatively large variety of mental models were used by students as shown by Figure 5. However, two mental models in particular attracted the most students – *NA* and *m2*. 32% of students utilised the m2 model, 88% of which used the *s3* juxtaposition. The large portion of student using s*3* combined with the fact that no students used *s1* suggests a misunderstanding of how variables are handled, supporting the idea that variables are a potential threshold concept for students however, the use of *m2* demonstrates that some students; regardless of background, can process compatible mental models which are appropriate for programming.

36% of students were categorized as *NA* meaning that they failed to use a known mental model or entered erroneous data. Interestingly, instead of inputting the value of the variables 4 students submitted code; such as variable names, as their answers. Only 21% of students consistently used a single mental model. However, due to high proportion of students using *NA* it is difficult to draw any statistically significant conclusions.

5.7. Predicting success of failure

At the beginning of a module teachers may not be fully aware of each student's programming ability. Whilst most introductory programming courses start from a basic level, some students, especially students who have attempted to teach themselves to program, may have already developed misconceptions about some of the fundamental concepts of programming, which staff would likely be initially unaware of. If these misconceptions are not identified and rectified by staff, students will construct new models of dubious quality based on their misconceptions [20], making the process of learning to program more difficult for the student, therefore increasing the possibility for failure within the module.

It is therefore important to identify students at the beginning of the module who are most at risk of failure, allowing staff to give them the support they need to overcome their misconceptions and any threshold concepts they may be facing.

The data collected during this research investigation has highlighted a number of factors that may impact on a student's performance within an introductory programming module, making it possible to split students into three distinct categories based on their mental model/juxtaposition usage:

On Track – Consistent use of *m2s1* (*m2s1 con*) indicates an appropriate mental model has been established, giving a student the best chance of success within the course.

At Risk – Students use *m2s1* as their dominant model, yet use it inconsistently (*m2s1 incon*) as students may be transitioning between mental models and may be experiencing cognitive conflict.

Students who use *m2s3* (either consistently or inconsistently) as their dominant model can also be categorised as 'At Risk', indicating students may be encountering a threshold concept that is blocking their progression within the module.

Students have the potential to succeed, however, they are at risk of being held back by threshold concepts and/or cognitive conflict. A student identified as 'At Risk' should be given specific help to ensure they overcome these issues by being encouraged to confront their misconceptions directly [12].

Falling Behind – Students use models other than *m2* either consistently or inconsistently (including *NA*), indicating they have not grasped the concept of how variables are handled and may potentially be experiencing cognitive conflict or being held back by threshold concepts.

These students are at the most risk of failure within the module, steps should be taken by staff to address their misconceptions directly [12], as well to help acquire appropriate domain knowledge in order for them to succeed in the module.

By splitting students into three separate categories teaching strategies can be adapted to provide material that is appropriate to each level, for example, students who are classed as 'At Risk' or 'Falling Behind' can be given material aimed at helping them overcome their misconceptions and any threshold concepts they may be encountering. If the same material was given to students who are classed as 'On Track' is possible they may become bored with the module, potentially leading to them underperforming.

The thought processes a student goes through when faced with a programming scenario is represented in the Programming Thought Process (PTP) Model. This model is an original contribution of this research and highlights how a student consults their domain knowledge before attempting to approach a programming scenario, which in turn influences the mental model and juxtaposition used by the student.

By also taking consistency into account, the PTP Model (Figure 6) provides a visual representation of how the thought process of a student relates to the previously described categories, thus allowing staff to identify where a student is encountering problems and provide appropriate support.

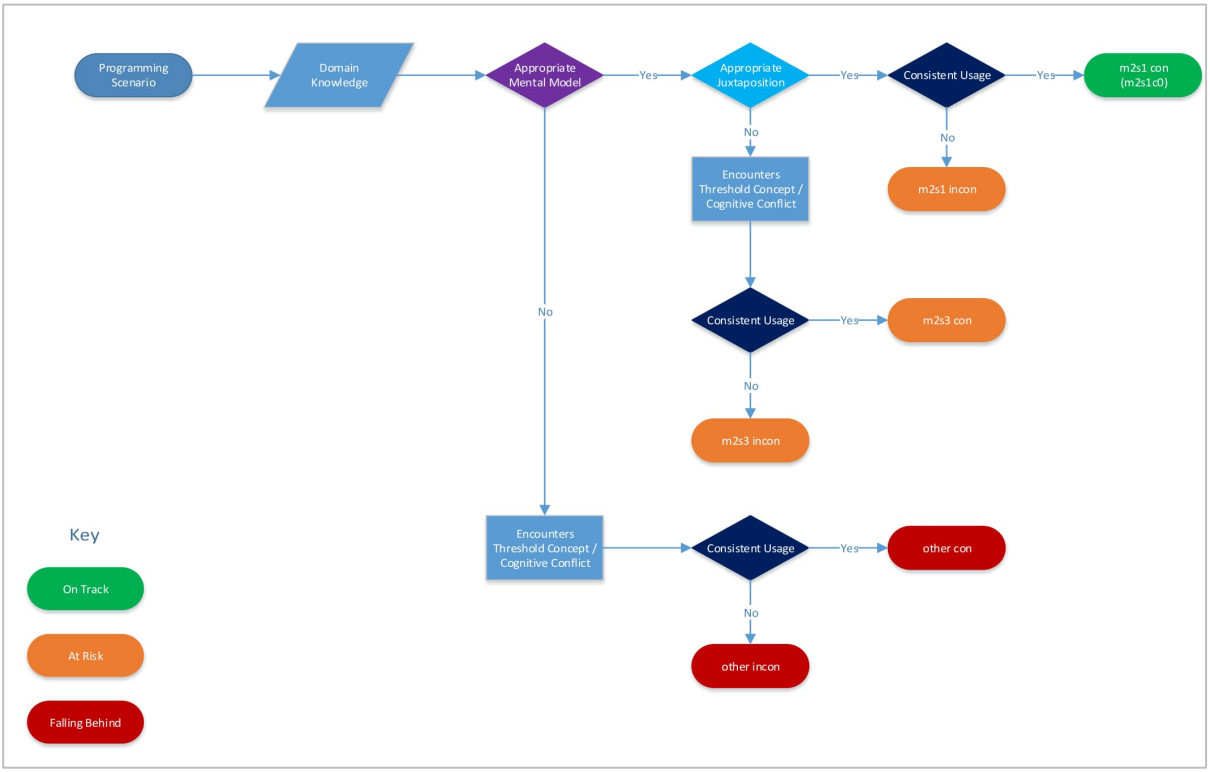

Figure 6. Programming Thought Process (PTP) Model

6. Conclusions and future work

We believe the following conclusions can be drawn from the data collected during this investigation:

- Consistent use of *m2s1* is a predictor of success within introductory programming courses.
- Variable swapping is a potential threshold concept for students; highlighted by the number of students using the *s3* Juxtaposition.
- As domain knowledge develops, students begin to gravitate towards appropriate mental models, however, they may encounter cognitive conflict or threshold concepts that can block their progress.

Despite being relatively successful, there is still future work required to validate the findings of this experiment and to further expand the understanding of the programming learning process.

By only collecting data over a single semester, the reliability and validity of this research are severely constrained. In order to gain a comprehensive understanding of the programming learning process that is valid and highly reliable, the same experiment should be run throughout the course of an entire year, over the course of multiple years and at various institutions.

Students studying a wider range of subjects; such as Media, Music, English Literature, and so on, should also be observed to determine how their mental model usage differs to Computer Science students, and investigate any potential related subjects in order to allow for a better understanding of students' domain knowledge.

Despite the limitations, we believe this research has highlighted the link between appropriate mental model usage and success within an introductory programming module. The data collected during this research has also revealed a potential link between related domain knowledge (from studying subjects such as Mathematics or Physics) and appropriate mental model usage, although further research is needed to validate these results.

7. References

[1] B. Adelson and E. Soloway, The role of domain experience in software design. *IEEE Transactions on Software Engineering*, (11), 1985, pp.1351-1360.

[2] M. Ben-Ari, Constructivism in computer science education, *Acm sigcse bulletin* 1998, ACM, 1998, pp. 257-261.

[3] N. Blackwood, Digital skills crisis: second report of Session 2016–17: report, together with formal minutes relating to the report: ordered by the House of Commons to be printed 7 Jun 2016.

[4] S. Dehnadi, Testing programming aptitude, *Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group*, 2006, pp. 22-37.

[5] B. Du Boulay, Some difficulties of learning to program. *Journal of Educational Computing Research,* 2(1), 1986, pp. 57-73.

[6] S. Furber, Shut Down or Restart: Report of the Royal Society into Computing in Schools, *The Royal Society, London,* 2012.

[7] Higher Education Funding Council For England and The Complete University Guide, Dropout Rates at English Universities.
Available: http://www.thecompleteuniversityguide.co.uk/news/dropout-rates-fall-at-english-universities/ (Access Date: July 30, 2016.

[8] P.N. Johnson-Laird, Mental models in cognitive science. *Cognitive science,* 4(1), 1980, pp. 71-115.

[9] O. Kerr and N. Danino, Programming Patterns as Potential Predictors of Student Success. *Proc. World Congress on Education (WCE-2017)*, 2017.

[10] E. Lahtinen, K. Ala-Mutka, and H. Järvinen, A study of the difficulties of novice programmers, *ACM SIGCSE Bulletin* 2005, ACM, 2005, pp. 14-18.

[11] G. Lee, and J. Kwon, What Do We Know about Students' Cognitive Conflict in Science Classroom: A Theoretical Model of Cognitive Conflict Process, 2001.

[12] S. Maier, Misconception research and Piagetian models of intelligence, *Proc. 2004 Oklahoma Higher Education Teaching and Learning Conf,* 2004.

[13] H.B. Mann and D.R. Whitney, On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics,* 1947, pp. 50-60.

[14] J. Meyer, and R. Land, Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines, *University of Edinburgh*, 2003.

[15] D.A. Norman, Some observations on mental models, *Mental models,* 7(112), 1983, pp. 7-14.

[16] M. Prensky, Digital natives, Digital immigrants part 1, *On the horizon*, 9(5), 2001, pp. 1-6.

[17] J. Rogalski, and R. Samurçay, Acquisition of programming knowledge and skills. *Psychology of programming,* 18(1990), 1990, pp. 157-174.

[18] M.A. Sasse, Eliciting and describing users' models of computer systems, 1997.

[19] Soloway, E. 1986 Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM*, *29*(9), 1986, pp.850-858.

[20] L.E. Winslow, Programming pedagogy—a psychological overview. *ACM SIGCSE Bulletin,* 28(3), 1996, pp. 17-22.