Object Oriented Programming and Program Correctness: The Students' Perspective

Ioanna Stamouli
Department of Computer Science
University of Dublin
Trinity College
O'Reilly Institute, Dublin 2,
Ireland
stamouli@cs.tcd.ie

Meriel Huggard
Department of Computer Science
University of Dublin
Trinity College
O'Reilly Institute, Dublin 2,
Ireland
huggardm@cs.tcd.ie

ABSTRACT

Many Computer Science and Engineering curricula contain core modules on computer programming and programming languages. An increasing number of institutions choose to introduce undergraduates to programming through object oriented languages. As part of a longitudinal phenomenographic study we have set out to investigate the understanding of programming concepts that first year undergraduate students have when learning to program and think in the object oriented paradigm.

The conceptions that students have developed on what learning to program really means and their perception of program correctness are explored; providing an insight into the levels of abstraction and complexity of the learners' understanding. Our findings suggest that the way students experience learning to program is related to their perception of what constitutes program correctness.

Categories and Subject Descriptors

D.1.5 [Programming Techniques]: Object-oriented Programming, and K.3.2 [Computers and Education]: Computer and Information Science Education.

General Terms

Human Factors, Languages.

Keywords

Phenomenography, Object oriented programming, learning to program, program correctness.

1. INTRODUCTION

Programming concepts and languages are fundamental for the study of Computer Science and it is critical that first year undergraduate students acquire the essential skills in these areas as quickly as possible. However, it is evident that many students find it difficult to acquire these skills and this has a negative impact on their performance throughout their undergraduate career [5]. In order to improve the quality of teaching and, therefore, the learning experience of undergraduate Computer Science students, it is essential to understand, how they experience learning to program and its associated aspects and features.

Based on these initial observations, we have undertaken a longitudinal study aimed at exploring the main conceptions undergraduate students have of the most fundamental principles of object oriented programming. In this paper we discuss a subset of our findings, examining the conceptions that novice programmers have of learning to program and their perception of program correctness within the object oriented paradigm. The data used to explore these themes is a series of interviews with sixteen first year undergraduate students. These were analysed following the phenomenographic research paradigm. This reveals the qualitatively different ways in which a phenomenon can be experienced, understood or perceived by a student cohort [8], and hence it is optimal for this study.

Analysis reveals the conceptions that students have developed of what learning to program really means, along with their perception of program correctness. Together, these results provide an insight into the levels of abstraction and complexity of their understanding. Our findings suggest that the way students experience learning to program is related to their perception of what constitutes a correct program.

In the remainder of this paper we present the details of the study and the chosen research methodology. In sections 3 and 4 the results are presented together with supporting excerpts from the interviews. The results are then discussed in section 5 where the relationships between the concepts and the implications of the findings are explored. We conclude section 6 by indicating possible directions for future research.

ICER'06, September 9–10, 2006, Canterbury, United Kingdom.

2. THE STUDY

The study was performed in the University of Dublin, Trinity College, over the course of an entire academic year. Sixteen first year undergraduate students were selected to participate in the study, drawn from the *Introduction to Object Oriented Programming* course that is part of the syllabus of the Computer Science degree. The programming language used to introduce the students to programming is Java.

In the following we present the reasons behind the selection of this course and student cohort. A brief overview of the research approach and the means used to collect the data in this project is also provided

2.1 Phenomenographic research

Phenomenographic research is strongly empirical in its essence. The first goal of a phenomenographic project is to describe the experience of learning *something* [2]. The particular object of the study constitutes the phenomenon under investigation; in this case learning object oriented programming and its specific concepts. The focus of a phenomenographic study is to describe the variations in the ways a phenomenon is experienced and understood within the population under investigation.

The results of such a study are presented as descriptive categories. Each category characterises a particular way of experiencing the phenomenon, capturing key aspects of the essential differences between the categories. The categories are usually presented in a hierarchical manner, since some describe a more advanced or complex understanding than others [1]. The variations between the categories present the critical points in understanding and are highlighted in the analysis.

The main method for gathering data when employing the phenomenographic approach is interviews. These are usually semi-structured in order to allow for flexibility in following interesting themes that may arise during the course of a session. The interviews are transcribed verbatim and analysed in detail. During the analysis the researcher is required to be open to alternative views of concepts. The material is analysed both as a whole, in the context of the interview, and also as individual statements, in the context of particular themes [1, 8]. This highly iterative process results in categories of description that capture the main conceptions by which the phenomenon is understood.

2.2 Selecting the Course and the Students

The focus of the project is on novice programmers' understanding of the concepts involved in object oriented programming. Thus the population most suited for the study is highly motivated Computer Science students.

The course starts as an introduction to object oriented programming using Java. Students require no prior knowledge of programming, or computing in general, in order to follow and complete the course.

Half of the course population participated in the study, which required their involvement in 4 paid hours of interviews distributed throughout the academic year. The students were selected based on a background questionnaire that was distributed at the beginning of the year. The criteria used for the selection of the students aimed to capture variations in prior programming experience, motivation for choosing the degree and previous academic performance.

2.3 Interviews

Although a series of four interviews was conducted with each student, the topics explored in this paper were discussed in the last two interviews. These were held during the second half of the course, towards the end of the academic year. The interviews were semi-structured; therefore some questions were prepared beforehand in order to provide an outline of the topics to be discussed during the session. The main questions that were used to investigate unveil the students' conceptions of learning to program were *"what do you think learning to program is all about?"* and *"what do you think it takes to learn how to program?"*. As regards exploring students' understanding of program correctness the questions that were used were *"when do you think a program is correct?"*, and *"what is your idea of a correct program?"*.

3. LEARNING TO PROGRAM

While learning to program is a theme that has been investigated in previous studies [3, 4, 6], these have been in educational disciplines rather than pure Computer Science. In this study we identified six distinct, and in some cases inclusive, ways of experiencing learning to program. These are presented in Table 1.

The first three categories focus on the programming language and its constructs; while the next two bring out the unique way of thinking that is required when programming. The last category describes how learning to program involves the acquisition of a new skill that can be utilised in everyday life. A more elaborate presentation of the categories follows, along with supporting excerpts from the interviews.

Category Label	Category Description
Learning the syntax of the language	Learning to program is experienced as learning the syntax of the language.
Learning and understanding the programming constructs	Apart from learning the syntax of the language, the focus here also includes learning and understanding the constructs involved in programming in general.
Learning to write programs	As above, but also utilising all these to write programs by having a structure in them.
Learning a way of thinking	In addition to the above, learning to program is also experienced as learning how to think logically in general.
Learning to "Problem Solve"	As above, and also utilising this way of thinking to solve programming problems.
Acquiring a new skill	The whole process is experience as learning a new skill that affects the way one thinks in real life.

Table 1. Categories of description for learning to program.

3.1 Category 1: Learning the syntax of the language

Students in this category experienced learning to program as learning the syntax of the language. The students that belong to this category seem to direct their efforts towards learning the keywords and the details of the syntax, in some cases even by heart. Knowing the details of the programming language provides the student with the ability to write a piece of code that compiles without any syntactical errors and this is what they experienced learning to program to be.

As Alan emphasises in his response, the important factor is knowing how to fix small, syntactical errors that may occur.

> Alan$_3$: Oh yeah, the syntax, you know, learning to fix the small errors that you would... you know, you have the basic knowledge but then you still need to refine it... you know, you may put the wrong brackets at some point and you may not know the equals method to compare the two strings, you know, stuff like that.

In his response to the questions, Alan refers to the basic knowledge one has about programming, but the main focus of his experience is on learning the grammar details of the language. Another student, Liam, points out that the only thing you need to do to in order to learn how to program is to get a book from which one can learn the syntax and features of the language.

> Liam$_3$: It's about getting a couple of books, really... Well it is the language that you are interested in anyway and the syntax of the language is what you learn from the book. Then you have to mess around with the syntax and the features of the language and having fun you learn more, really […]. So what you can do depends really on the language and this is, really, what you learn from a book, the syntax of the language, that's all you need anyway.

Here we get the perceptions of a complete novice (Alan) and a relative experienced programmer (Liam) where the focus of their experience of learning to program is primarily on learning the syntax of the language, independent of their prior programming experience.

This first, basic category of what learning programming is about, has been also identified in other studies [3, 4, 6].

3.2 Category 2: Learning and understanding the programming constructs

Students in this category are focused on the process of learning and understanding the constructs involved in programming in general and, thus, follows directly from the first category. The understanding expressed in category 1 is presumed, however the learner views the syntactical details to be in the background of their experience. The dominant feature of this category is understanding the essence of the programming constructs. Neil expresses this conception strongly:

> Neil$_3$: [...] you need to know the syntax of the language that you are programming in and the concepts like iterations and conditions. But all these… you kind of need to know how to use them, like, and when to use them... The syntax of these things is also important but you need to understand the concepts first not in a single language, like, because they kind of exist for most of the programming languages.

Neil is very clearly distinguishing between the syntactical details of the language and the programming concepts such as iterations, conditions, etc. Stephan also talks about the techniques you need to learn when programming. When asked what it takes for someone to learn how to program, he replied:

> Stephan$_3$: It takes… A good deal of work actually.
>
> I: Work on what?
>
> Stephan$_3$: On studying concepts… programming concepts and the languages.
>
> I: When you say concepts what exactly do you mean?

Stephan$_3$: Well, basically, techniques, the basic understanding of how to make a computer do something and then the language is something that you learn in order to translate that to something that the machine can understand, but you have to understand first how the concepts work so you can use them when you need to the right way. You know...

This category of description was also found amongst the sample population in study [4].

3.3 Category 3: Learning to write programs

In this category the focus moves from the syntax and the programming constructs to the actual writing of programs. Thus learning to program is experienced more as utilising the syntax and the features of the language to write programs that solve the problem at hand. Although the focus is clearly on the programs; the nature of the programs is not specified. Anthony exhibits this conception in the following answer:

I: What do you think it takes to learn how to program?

Anthony$_3$: It involves setting out your ideas after thinking how to, like, put these ideas into the proper syntax and you need to know the commands in whatever language you are working in and then writing out the program. That's about it really.

Anthony's conceptions include the understanding presented in categories 1 and 2, however the emphasis is on how one should use all these to write programs.

Sean points out the importance of learning from other programming examples:

Sean: Ehm... first of all the basics of how the language works and what is the grammar that it uses... and then just various different examples, just to get used to how it is being used really. That is the only way to learn it really properly by writing and writing... small programs at the start and then bigger as you get to know more. You need lots of experience.

Booth [2] also identified this category in her investigation of learning to program.

3.4 Category 4: Learning a way of thinking

In this category the perception has progressed from the language and programs to learning a new "way of thinking". Many students expressed that, when programming, one needs to be in a different frame of mind than that needed for other courses. Hence, in order to be able to program one needs to learn to think in this new way. This frame of mind has been vaguely described as *logical thinking* by the students in this study. Patrick explains this it in the following way:

I: What do you believe is required, or what does it take to learn how to program?

Patrick$_3$:You definitely need the sense of logic, that is the basic thing because everything else then is based in your ability to logically think of a solution, even with some of the other programs that we are doing you need the logic to be able to see it. Basically, you need decent mathematics and a basic sense of logic and the practice, of course. Pay attention and then it clicks, I guess.

Patrick here links the process of learning how to program with learning mathematics. This may have been influenced by the fact that the students in the labs and tutorials were given programming examples and problems based on mathematics. One thing is clear though: this category reveals that programming requires the learner to be in a different frame of mind where logic is central. Patrick mentions something else as well in his response, this "click" moment that many students talked about in our informal conversations during the course. This moment is described as something that one cannot control and when it happens, programming starts to make sense and the learner is able to program as if all the knowledge has fallen into the right place. A similar conception to this has been documented in [6].

3.5 Category 5: Learning to "Problem Solve"

This category derives from the previous one in that the concept of logical thinking is now expressed as a condition for problem solving. This more complete conception involves learning to problem solve as part of the experience of learning to program. Being able to see the solution to a problem is a critical feature of this category. The importance of problem solving is highlighted by the teaching staff from the very beginning of the course and the students have been taught to perform case analysis when attempting to solve a problem. This involves finding all the different cases that may arise in the solution to a problem and the conditions that affect them. Usually, students are encouraged to write down the conditions in English and use this as a guide when writing the code for their solution. This has possibly affected the number of students who belong to this category. For example, Eamonn describes this in the following quotation:

Eamonn$_3$: [...] Ehm logic, how to think logically and... breaking the whole thing down. Like in the tutorial the demonstrator told me that you need to break the whole thing down and then you work what you need to have and how you go about it [case analysis], and then the whole thing made a lot more sense. It is just ... you only understand by being shown things... the book is good so then you only have to learn it [the syntax] off […] Let just say... I am actually coming from the... actual point that Java code doesn't matter that much, like I haven't learnt it. Like I can do it with a book along side I am not confused anymore. But the important thing is to think is

this way so you see the solution... Do you know what I mean?

Unlike Eamonn, Brian does not explain how he learnt to solve programming problems; rather he focuses on the characteristics that a good solution should have.

Brian$_3$: I think it's more trying not to look at it as one as more like not just as the problem, but more as to how you are going to deal with the problem. There is a lot of ways probably... an infinite number of ways that you can do any one problem. It's just trying to do it the smartest and quickest way really… not the quickest way, the best, the more efficient way.

Declan points out the difference between problem solving in programming and problem solving in other situations:

Declan: Hmm logic and just kind of… hem… it has to be clear in your mind what you have to do, if you know what I mean, like,[…] there is always going to be certain cases when you have to solve a problem and you just have to know what rules you have to follow and what kind of things you have to do to solve it. […] because you can solve problems without having to think so much about it, where in computer programming solving a problem is more complicated because the instructions are so basic you have to go down and solve the problem into its very basic elements, like, think… a lot of the problem and its solution. Because when you are thinking of a problem solution in your mind you take a lot of things for granted but it is not this way in programming, because you have to think more basic.

The focus in this category is on problem solving, and the students have described how one can learn this technique. This category reveals a more developed conception of learning to program that has extended beyond the previous categories to include the problem as well as the ways one can approach its solution. The logical way of thinking that was described in category 4 is now used to enable problem solving. This is a similar finding to that presented in [6].

3.6 Category 6: Acquiring a new skill

The final category conceptualises learning to program as learning a new skill that affects the way one thinks in real life. This conception assumes understanding of the language and the programming constructs while it draws on elements from categories 4 and 5. However, the way that thinking logically enables problem solving is now viewed from a different perspective that extends beyond the course to the real world. Students experience learning to program as the acquisition of a new skill that can be useful in other areas of their studies, such as learning other programming languages, or even in different areas, like mathematics. Colin voices this understanding in the following:

I: Have you learned anything else to reach that point in programming ability that you have now?

Colin$_3$: Well I didn't have to carry out anything very difficult so far but, yeah, you learn how to break down things to more manageable pieces which would be applicable… to any programming languages and any mathematical operations or anything really.

In a different interview session when students were asked what they thought was the most important thing they learned during the course, Ken and Cormac said the following:

Ken$_4$: [...] sort of you work out how to analyse problems and you realise that maybe there are ways of doing something, that you haven't thought of before and you can apply that to other things like in assembly.

He then continues by giving a mathematical example where the logical thinking he learned in the programming class helped him solve in his mathematics class. Ken has explained how his new skills helped him in other courses while Cormac moves even further:

Cormac$_4$: Yeah I did (learn) definitely logic and problem solving etc. and it helps in many other ways like cleaning the flat (laughs)

I: How was that?

Cormac$_4$: (giggles) I mean the first time we tried to clean it everyone was trying to do the same thing, but the next time we said, you do this and then you do the other, and everything happened very fast and very structured.

Cormac argues that learning to program has affected the way he behaves and accomplishes tasks in real life situations, like when he is cleaning his flat. This conception removes the experience of learning to program from the strict boundaries of the course and university, and makes it part of the learner's everyday world.

4. UNDERSTANDING OF PROGRAM CORRECTNESS

An understanding of correctness is an important aspect of the experience of learning to program since it affects the approach a learner adopts. This theme was taken up in the last interview session. This took place after the students had received feedback on almost all their assignments and laboratory work and at a point where their understanding of programming was reaching a more mature stage. Students' understanding of program correctness has been known to differ from professional or academic standards [7] and this is evident in our categories.

Category Label	Category Description
Syntactical correctness	A program is perceived to be correct when it is syntactically right, that is when it compiles without any errors.
Functional correctness	Apart from being syntactically correct the program needs to fulfil the requirements of the problem specification.
Design correctness	In addition to the above, the program should be correctly structured in order to enable extensibility.
I/O validation and performance correctness	As above and, also, the program should cater for invalid input and it should also be optimised in terms of code length and how fast it executes.

Table 2. **Categories of description for understanding of program correctness.**

Four qualitatively different ways of experiencing "correctness of a program" have been identified: the first two categories focus more on the problem and the code correctness of the program while the last two are more developed, since they incorporate non-functional elements that aim to assist the actual user of the program. The categories are summarised in Table 2 and these are further analysed in the sections that follow.

4.1 Category 1: Syntactical correctness

In this category a program is experienced to be correct when it is free of any syntactical errors, in other words when it runs. The student focus in this category is solely on the code. Neither the problem requirements nor the human aspect of programming are involved in this perception. The experience is narrowed down to the relationship between the code and the programmer, as Liam explains in the following:

I: When do you think a program is correct?

Liam$_4$: When it had no bugs.

I: Anything else?

Liam$_4$: No, not really, as long as it runs, it's right.

Alan, in the next quote, elaborates a bit more by saying that a program should use all the methods and classes that he has previously defined. However the focus of correctness remains the syntactical correctness of the program itself.

Alan$_4$: When it works and the application runs and calls all the methods in the class and everything is fine and no errors. […] I find that doing it on paper first is easier for me.

The understanding of when a program is correct in this category is restricted to the code and the language that is used. The student is satisfied with the correctness of the program when it runs, independent of the functionality of the program. Fortunately, not many students within the participating population shared this view.

4.2 Category 2: Functional correctness

This category of description expresses an understanding of program correctness where the problem requirements are the focus. The understanding that was expressed in the previous category is still present; however the central point of the experience has expanded beyond the code to include the problem definition and the requirements of the application.

As Patrick explains in the following statement, program correctness is about satisfying the requirements of the problem:

Patrick$_4$: I suppose when it satisfies all the things in the question and when it compiles successfully or when the red lines disappear from Eclipse [laughs]. I suppose it is right when it does what you want it to do.

Another opinion is expressed by Stephan in his final interview:

Stephan$_4$: Well I suppose that the way I look at correctness is a bit different than what the lecturer believes. He thinks that a program is correct only if it follows the object oriented way while I think that a program is correct when it does what it has to do. Also the source code has to look correct and readable.

The lecturer has being emphasising the importance of following object oriented design principles when developing an application. However Stephan, who has prior programming experience with procedural languages, cannot see the point in object oriented design as long as the necessary functionality is implemented; design is not considered to be part of a program's correctness here. Students in this category experience program correctness as being related to the code being error free but the primary focus is on the problem definition rather than the code *per se*.

4.3 Category 3: Design correctness

In this case learners experience program design as the principal criterion for correctness; affecting the extensibility and readability of the program. Therefore, apart from all the other properties that should be present in a solution, the right structure of the program is a key issue. Eamonn explains this as follows:

I: I mean let's take for example your poker program, which were your criteria of correctness?

Eamonn$_4$: I felt it was correct when it runs, did what it was supposed to do and it is structured properly. [...] I feel design is part of correctness. It is easier and better when it is correctly structured, because people can understand it... and because then you can go back and extend and reuse what you had there. So appropriate design is really important and it adds to the solution.

Brian explains how his perception of when a program is correct has changed after the exam.

Brian$_4$: I used to just say when it does what is supposed to do but it was... Karl, when we had that sort of half exam just before Easter and it [his program] was right I mean what he wrote was right but it was missing something small and for that he got I think 10 out of 40 or something, so it is kind of like... it kind of has to be functional as well it has to be open and object oriented and it has to be sort of modular [he writes down again] because he had it all in big blob of code. So it is right if it is like... I don't like to think that... if the final result is right so the whole thing is right. It is how you got there, how you designed the thing to get there as opposed to correctness as such.

Brian says that he previously believed that program correctness was all about the functional elements of the program, but as his friend had marks deducted from his exam paper because his design was inappropriate he changed his mind. Brian's focus in his answer is clearly on design and this view is unlike the previous categories. In the last sentence of his response Brian makes a distinction between the end product correctness and the process that one follows to achieve that. Thus, from his point of view what is important is the techniques used to develop a solution and that this is what constitutes the program's correctness. Declan is thinking along the same lines in his response:

Declan$_4$: Well, if it works firstly and then the way the code is written and structured because sometimes code can look terrible, I mean really horrible, like if you got like i+= 1 don't like it because it is very hard to read. Just, generally on how easy the code is to read and so on...

Although all of the students quoted above experience design to be an important part of correctness, their motivation is very different. Eamonn stresses the importance of design because that makes the solution easier to extend, Brian is concentrated on achieving good grades and separates syntactical and design correctness, while Declan points out that the right structure improves readability and therefore makes the program correct. These students share the same perception of program correctness, however they approach it from very diverse and different angles.

4.4 Category 4: I/O validation and performance correctness

In this category the perspective is broadened even more to include non-functional requirements as part of the experience:

Anthony$_4$: When it fulfils the functions that it supposed to do... without any side effects, it might be able to fulfil all the tasks but it should be responding in cases where the user enters something invalid.

I: You mean checking for the validity of the user input?

Anthony$_4$: Yeah.

Anthony here emphasises the importance of input validation and he experiences this as being part of a correct program. Thus, apart from the functionality that should be implemented, the learner considers non-functional elements such as error checking to be necessary when a solution is developed. Even though the focus is still on the problem requirements, there is more to correctness than just the basic functionality. Colin expresses this more strongly:

Colin$_4$: A program is correct when it does what you want it to do first of all, so you give it the values you want it to use and then it just works. It works also when it is user proof so if you use the wrong values then you cannot crash it, you cannot pass values that would make it not work. You have to be able to respond properly when you don't give exactly what it wants. It should be able to distinguish among what is valid and what is not. It should also be what is the word... optimised. It has to be as short as possible to do it and it should also take up less memory and it should run faster.

I: Did you came up with this by yourself, or you read that somewhere?

Colin$_4$: Oh, by myself, from my experience.

Colin's understanding of correctness takes into account the behaviour that a program should have when it is executed. From his point of view the responses a program provides to the user should be meaningful. Hence, in order for a program to be correct, input and output validation is necessary. Colin also emphasises other non-functional properties such as code length and memory efficiency. Kevin emphasises the importance of providing clear guidelines regarding the user input and program usage:

Kevin$_4$: When you run it and it does everything you kind of ask from it to do, say it looks for user errors, so if you ask for yes or no and you expect the user to type 'y' or 'n' then if the user types something else this would not crash the program, like, it will deal with that correctly. So sort of input and error checking is important as well.

Mark thinks along the same lines in his response:

> $Mark_4$: When it does what it is supposed to do without errors I suppose hmm… Caters for any error that might occur any problems like… yeah problems that might arise form the user doing something that he is not supposed to rather than crashing and it is supposed to do whatever you expect it to do rather than something that you either don't need or don't want. It is correct when it solves the problem and prevents other problems from occurring.

All the above quotes reveal an understanding of correctness that incorporates the previous categories but focuses mostly on I/O validation and other non-functional properties such as optimisation and efficiency. The focus of the students' experience has expanded to involve the user as well, since the importance of validating input and output in this category aims to assist the user in using and acquiring a better understanding of the application.

5. DISCUSSION

In this section the structural aspects of the categories of description for the two themes (Learning to program and the understanding of program correctness) are discussed and the relationship between them is examined.

5.1 Findings on Learning to program

The qualitatively different ways that students experience learning to program when they are first introduced to it, are presented in Table 1. The six categories that have been identified within the study's population are clearly distinct and inclusive, in the sense that each one assumes the conception(s) formulated in the preceding categories. Thus, the earlier categories express a relatively basic understanding and, as we progress through the categories, the experience becomes richer and the view broadens and matures.

The first three categories, *learning the syntax of the language*, *learning and understanding the programming constructs* and *learning to write programs* reveal an understanding that is more strongly oriented towards the technicalities of the process of learning to program. However the focus of each category is different. The experience in the first category is solely limited to the student learning the syntactical details of the underlying programming language. In category 2, the conception has evolved to a more abstract understanding of programming constructs and the experience broadens further in the third category where the focus is on the way all of the previous categories enable the creation of programs. The existence of these particular conceptions is not surprising as they fit, more or less, within the structure and development of the course. In particular, the study of programming focuses on each of the elements present within the first three categories at different stages of the course presentation. However, since the interviews that discuss this theme were held during the latter stages of the course it was hoped that the students would have moved from this limited conception to a more mature experience, such as the ones described in the next three categories.

The next two categories, *learning a new way of thinking* and *learning to 'Problem Solve'*, encompass a view that is not intrinsic to the course content or structure. In category 4, this way of thinking is expressed as logical thinking. Students that belong to this category have realised that there is something more to learning to program and that it is about learning a systematic or logical way of thinking. Hence, the students felt that this is what one should learn, and concentrate on, when learning to program. However, the conception described in category 4 is not complete, since this 'way of thinking' is not clearly understood and the students refer to a "click" moment, where all prior knowledge falls in place. In the penultimate category the conception has become clear: learning to program is experienced as learning to solve problems using programming constructs and techniques. The conception has moved from the detail of the programming language to a more abstract understanding that is centred on the actual problem. Although this view was not explicitly part of the course, it was encouraged and fostered throughout the course, especially when programming tasks were assigned to the students.

In the final category learning to program is viewed as acquiring a new skill. This conception presupposes the aspects discussed in the previous categories and concentrates instead on something external to the components of programming. This complex conception views the process of learning to program as learning a new skill that is not only required when programming and solving problems but one that changes the way a person thinks and acts in life. This structured and logical way of thinking, which was initially outlined in category 5, is now put in the context of the learner's everyday life. Those in this category have discerned the skills that are inherent in learning to program and are now using them in other courses or activities, such as in solving mathematical problems, learning other programming languages or everyday activities that require a structured way of thinking.

5.2 Findings on Understanding of Program Correctness

Four distinct categories of description were identified in the conception of program correctness theme. The students were asked what constitutes program correctness within the context of the course rather than in general. Thus the qualitative categories of description were formulated with this in mind. The first two, *syntactical* and *functional correctness* reveal a conception that is

focused on the more tangible elements of the theme. Category 1 *"A program is perceived to be correct when it is syntactically right, that is when it compiles without any errors"* describes a conception where the central focus is the programming language. Students that share this viewpoint have difficulty understanding programming and in many cases remain puzzled when they do not achieve the grades they expected.

Functional correctness was a popular category among this study's population. The fulfilment of the problem requirements is the focal point of this conception. Even though this is a straightforward and very logical conception, students that experience program correctness in this way often fail to achieve their potential in a course such as the one under investigation. When learning object oriented programming, an essential goal of the course is for the students to learn to write and think according to the object oriented paradigm. Not taking object orientation into account results in incorrect or incomplete solutions. From the analysis of the interview data, it appears that students with previous experience of procedural programming languages failed to see the importance of the object oriented paradigm.

The next category presupposes the understanding expressed in the previous two conceptions and instead focuses on the design of a program. Students that share this view do not merely try to fulfil the requirements of a given problem, but try to follow object oriented techniques and develop an extensible and more reliable solution. This conception reveals a richer understanding of what constitutes a correct program. This is clearly differentiated from the previous category and has been placed in a more realistic paradigm where programs can be reused or further extended.

Finally the category labelled *"I/O validation and performance correctness"*, focuses mostly on the interaction between the program and the user. The learner has moved beyond the tangible elements of correctness and is now concerned with the performance aspects of the program. The actual user is central to this conception and this enables the students to experience a correct program as something that solves real world problems and therefore should be designed to interact appropriately.

5.3 Relationship between the Themes

Our findings suggest that the way students experience learning to program is related to their perception of what constitutes a correct program. Students whose experience falls into the first four categories of description for what it means learning to program (*Cat. 1: Learning the syntax of the language, Cat2: Learning and understanding the programming constructs, Cat. 3: Learning to write programs, Cat. 4: Learning a way of thinking*), experience program correctness as either syntactical or functional. Although a one-to-one relationship was not established between the two themes: from the 9 students that fell into the first four categories of the first theme, all, except one, viewed program correctness as syntactical or functional. Thus, the way one experiences learning to program influences how one understands the outcome of that process, which i.e. the program itself. In the first four conceptions of the learning to program theme, the experience is confined to the programming language and the programs. Similarly, for the program correctness theme, the first two categories focus on the language and the problem at hand, rather than taking the bigger picture into account.

For categories 5 and 6, (*Learning to 'Problem Solve'*, and *Acquiring a new skill*) a one-to-one relationship is observed with *design and I/O validation* and *performance correctness*. The distribution is that from the 7 students that belong to the aforementioned conceptions, 3 experienced program correctness as *design correctness* while 4 experienced programming as *I/O validation and performance* respectively. Thus, when the focus of learning to program is on the structured way of thinking that enables problem solving, the primary criterion for program correctness is the object oriented structure and extendibility of the solution. Finally, when learning to program is conceptualised as the process of acquiring a new skill that can be used in real life; the understanding of program correctness is focused on the interaction between the user and the program, taking into account non-functional elements such as optimisation of the code and performance of the program as a whole. Hence, the results suggest that there is a clear linear relation between the two themes discussed in the paper.

Although, the study's sample population is not sufficiently large to draw final conclusions on the relations observed, it is sufficiently representative to indicate the existence of this trend. The results suggest that students develop a general view about learning programming and the programming constructs and that this then influences their experience throughout the course and even, maybe, their undergraduate career.

The findings also show that more than half of the population of the study does not develop a complete and mature understanding of learning to program such as the ones described in categories 5 and 6. A similar relationship holds for the understanding of program correctness. The interviews were held almost at the end of the academic year, so there were some students who completed the programming course and who still believed that learning to program in the object oriented paradigm is purely about learning the syntax of the language and that a program is correct when it is free from syntactical errors.

Undoubtedly the desired course outcome is for students to develop a deep understanding of programming and the

programming constructs, enhancing their experience and putting the knowledge into context. The question however is how educators can achieve this. Our findings suggest that there is a linear relation between the two themes, which begs the question: does this hold true for the students understanding of other object oriented constructs such as objects and classes? It would also be beneficial to know if by positively influencing one theme (such as the students' perceived criteria for program correctness), does this have a concomitant impact on the students' experience, and attitude, towards programming? These are questions that our ongoing research aims to explore.

6. CONCLUSION

This paper investigated the understanding first year undergraduate Computer Science students have of what it means to learn how to program within the object oriented paradigm. It also explored the students' understanding of what constitutes a correct program. Analysis of the data suggests that more than half of the sample population did not reach a mature stage of understanding of what it means to learn how to program. Similarly most of the students' experience of program correctness focuses mainly on the tangible elements of the language and the problem, instead of viewing correctness in context. Our findings also suggest that the way students experience learning to program is related to their perception of what are the criteria for program correctness.

It is essential that educators are aware of how students experience and understand specific programming constructs and programming as a whole. This study not only provides a deep insight into the students' understanding of these themes; it also provide a strong foundation for future work aimed at helping students' reach a more mature understanding of object oriented programming.

7. ACKNOWLEDGMENTS

This project is funded by the Irish Research Council for Science, Engineering and Technology, (IRCSET) under the Embark Initiative, and by the Information Technology Investment Fund, administered by the Higher Education Authority, Ireland.

8. REFERENCES

[1] Akerlind, S. G., Principles and Practices in Phenomenographic Research, in *Proceedings of the International Symposium on Current Issues in Phenomenography*, Australia, pp. 1-17 2002.

[2] Booth, S., A Study of Learning to Program an Experiential Percpective, *Computers In Human Behaviour*, Vol. 9., pp. 185-202, 1993.

[3] Booth, S., *Learning to Program. A Phenomenographic Perspective*. No. 89 in Goteborg Studies in Educational Science. Acta Universitatis Gothoburgensis, Goteborg, Sweden, 1992.

[4] Bruce, C., McMahon, C., Buckingham, L., Hynd, J., Roggenkamp, M. and Stoodley, I., Ways of Experiencing the Act of Learning to Program: A Phenomenographic Study of Introductory Programming Students at University. *Journal of Information Technology Education*, Vol. 3, pp. 143-160, 2004.

[5] Carter, J. and Jenkins, T., Gender and programming: What's going on?, *SIGSCE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE*, pp. 1-4 Poland, 1999.

[6] Eckerdal, A. and Berglund, A., What Does It Take to Learn 'Programming Thinking'?, in *Proceedings of the 2005 International Workshop on Computing Education Research, ICER'05*, pp. 135-142, Washington, 2005.

[7] Kolikant, Y. B. D., Students' Alternative Standards for Correctness", in *Proceedings of the 2005 International Workshop on Computing Education Research, ICER'05*, pp. 37-43, Washington, 2005.

[8] Marton, F. and Booth, S., Learning and Awareness. Lawrence Erlbaum Ass., Mahwah, NJ, 1997.