Received: 11 February 2020 | Revised: 10 August 2020 | Accepted: 17 August 2020

DOI: 10.1111/jcal.12498

ARTICLE

Journal of Computer Assisted Learning WILEY

Threshold concepts, conceptions and skills: Teachers' experiences with students' engagement in functions

Maria Kallia | Sue Sentance

School of Education, Communication & Society, King's College London, London, UK

Correspondence
Maria Kallia, School of Education, Communication & Society, King's College London, Waterloo Bridge Wing, Franklin-Wilkins Building, Waterloo Road, London SE1 9NH, UK.
Email: maria.kallia@kcl.ac.uk

Funding information
Google

Abstract

Threshold concepts have been characterised in the literature as jewels in the curriculum as they can inform teaching and learning practices. Therefore, identifying and addressing threshold concepts in any discipline is critical. The aim of the current study is to explore the existence of threshold concepts in computer programming and specifically with regard to the area of functions. Based on our previous works in which we identified 11 potential threshold concepts in functions by employing the Delphi method and seven misconceptions that students hold in this area of programming, the current study further explores computing teachers' experiences with students' engagement with 4 of the 11 concepts using an interpretative phenomenological analysis of interviews. The analysis revealed that from these concepts, we could argue that parameters, parameter passing and return values likely form a threshold conception and procedural decomposition is a procedural threshold (threshold skill). The study presents our framework that lead us to the identification of these thresholds in computer programming, presents the computing teachers experiences with these concepts and concludes with the implication of these results on students' learning and teaching practices in computer programming.

KEYWORDS

computer programming, pedagogical issues, teaching/learning strategies, threshold concepts

1 | INTRODUCTION

In recent years computer programming has become part of the curriculum in secondary education in many countries, indicating that programming skills are becoming more and more important and potentially a core skill. The importance of computer programming as part of the school curriculum and its relationship to students' cognitive skills has been investigated by many researchers (Clement, Lochhead & Soloway, 1990; Taylor, Harlow & Forret, 2010; Pardamean, Honni & Evelin, 2011; Fox & Farmer, 2011; Psycharis & Kallia, 2017; Tu & Johnson, 1990; Popat & Starkey, 2019). However, computer programming is difficult (Yizhou & Lehman, 2017) and as part of the computer science curriculum in many countries introduces many challenges both to students and teachers. Specifically, students may have difficulties understanding concepts and this makes the teachers' role very important. Threshold concepts, as a source of troublesome knowledge, play a significant role in these problems; both identifying threshold concepts and developing effective learning and teaching methods for programming are increasingly important in school computing education.

Understanding the characteristics of threshold concepts can significantly impact curriculum structure and design and the creation of powerful learning and teaching environments. That is why they are often called the *jewels in the curriculum* (Land, Cousin, Meyer & Davies, 2005). In this paper our starting point is that identifying threshold concepts in a discipline is imperative; they can first be used as a diagnostic tool to emphasise curriculum areas that need special attention from the teachers' perspective (Akerlind, McKenzie & Lupton, 2010), and second, they signify parts of the curriculum where students experience problems and are confronted with troublesome knowledge (Perkins, 1999).

Guided by the importance of identifying threshold concepts and the limited work that has been done in this area in computer programming, this study presents computing teachers' experiences teaching specific concepts that we had previously identified as potential threshold concepts.

1.1 | The current study

The study that is presented in this paper is part of a larger research project which aims to identify threshold concepts in secondary computer programming with particular regard to functions, and to subsequently make pedagogical and didactical suggestions that would guide computing teachers and promote students' learning. In this paper, we address the first part of the research goal: the identification of threshold concepts in the area of functions, using teachers' experiences as a lens. This particular study is based on our previous study (Kallia & Sentance, 2017) and extends it by shading lights into this complicated phenomenon.

Our previous research led to a set of *potential* threshold concepts and was based only on suggestions of computer science teachers without a concrete justification or deep investigation of the reason why these concepts appear to be thresholds to students' progress. In fact, what we observed was that teachers tended to suggest concepts based mostly on the troublesome aspect of these concepts and not the integrative and transformative nature of threshold concepts, two of the most critical characteristics of the threshold concept framework. Thus, our previous work only resulted to a list of concepts that can potentially be considered as thresholds but further research needs to be conducted and specifically explore the transformative and integrative aspects of these concepts.

With this research study, we explore further this phenomenon by delving deeper into computer science teachers' experiences teaching these concepts. The main aim of this investigation is to provide empirical evidence that supports the argument of these concepts being threshold concepts in programming. To achieve that we need to identify what constitutes these concepts to be perceived as thresholds by the teachers and provide links between these experiences and the threshold concepts characteristics. Particular emphasis is given in revealing the transformative and integrative characteristics of these concepts; uncovering these characteristics would be the strongest indication that the suggested concepts are actually thresholds in computer programming.

This study extends our previous work and, by interviewing computing teachers and by employing an interpretative phenomenological analysis (IPA), presents the results relating to four of the eleven concepts identified as potential threshold concepts: parameters, parameter passing, return values, procedural decomposition. The study did not investigate the concepts of abstraction and recursion as other researchers have already investigated these concepts (Male & Baillie, 2011; Boustedt, Eckerdal, McCartney, Moström, Ratclfie, Sanders & Zander, 2007; Eckerdal, McCartney, Moström, Ratclfie, Sanders & Zander, 2006; Holloway, Alpay & Bull, 2010; Rountree & Rountree, 2009). For the remaining five concepts, not enough evidence was found from the teachers' interviews that can support their nomination as threshold concepts, and they were therefore not included in this paper.

Therefore, the research questions of this study are as follows:

- What are computing teachers' experiences with respect to the teaching of parameters, parameter passing, return values and procedural decomposition?
- Is there evidence that supports the nomination of these concepts as threshold concepts, skills or conceptions?

The contribution of this paper can be summarised as:

1. A thorough interview data analysis (IPA) with computing teachers on their experiences teaching the concepts of parameters, parameter passing, return values and procedural decomposition.
2. A qualitative study which considers the threshold concept framework, acknowledges the aforementioned findings, and draws the conclusion that the group of parameters, parameter passing and return values together seem to form a threshold conception in computer programming and procedural decomposition is possibly a procedural threshold (threshold skill) in computer programming.

To the authors' knowledge, this is the first study that makes these suggestions and supports them with empirical data. Our study's findings have direct implications in the way teachers should teach these concepts and design curriculum materials.

2 | CONCEPTUAL FRAMEWORK

2.1 | Threshold concepts

Threshold concepts were introduced by Meyer and Land (2003) as a result of research in the UK on the characteristics of robust teaching and learning environments at undergraduate level. Meyer and Land suggested that certain concepts, inside every discipline, are fundamental to the discipline's mastery and that their understanding is critical for making progress. They define these concepts as thresholds because once understood they lead to new conceptual understandings that were previously hidden (Meyer & Land, 2003). Specifically, a threshold concept is considered as "*akin to a portal, opening up a new and previously inaccessible way of thinking about something. It represents a transformed way of understanding, or interpreting, or viewing something without which the learner cannot progress*" (Meyer & Land, 2003, p. 1).

Threshold concepts share some common features. Firstly, threshold concepts delineate the boundaries of particular disciplines which means that they indicate specific understandings that are characteristic to a specific disciplinary discourse (Wright & Hibbert, 2015). Second, threshold concepts are troublesome as they are conceptually difficult to be understood and third they are irreversible, as once understood it is difficult to be unlearnt (Rountree & Rountree, 2009). However, the principal features of threshold concepts stem from their transformative and integrative nature and it is these characteristics that make them different from other concepts in a discipline like core or fundamental concepts (Meyer & Land, 2003). Their transformative nature causes a shift of the students' understandings and insights of the subject or a part of it; this change may be attributed to their integrative nature which discloses conceptual connections and their interrelatedness (Cousin, 2006; Sandri, 2013).

For students, to experience such transformative and troublesome concepts is tough, and for that reason they often find themselves struggling to understand them. Learning becomes a barrier (Perkins, 1999). Students get stuck in these curriculum areas and are unable to pass through these conceptual blocks. This transition period is known as a liminal space in the threshold concept framework and is described as an insecure space where students' learning and understandings fluctuate between the known and the unknown, where students' understandings is based on mimicry and where uncertainty, anticipation and anxiety are emotions that delude students' learning experiences (Eckerdal, McCartney, Moström, Sanders, Thomas & Zander, 2007).

There has been quite a large amount of literature focusing on identifying threshold concepts in a variety of disciplines. For example, Davies and Mangan (2007) investigated threshold concepts in Economics, Male and Baillie (2011) in Engineering, and Cook-Sather (2014) investigated threshold concepts in academic development and found that student-faculty partnership is a threshold concept. Blackie, Case and Jawitz (2010) suggested that student-centred teaching is a threshold concept while Gourlay (2009, p.189) uses the term threshold practices to emphasise the interplay between "(a) the indeterminate, tacit nature of academic writing; (b) the emotional and social dimension of the student transition; and (c) the role of struggles around writing in identity formation". In computer programming, the most concrete work has been conducted by Eckerdal and Thuné (2005) and Anna Eckerdal et al. (2006, 2007).

2.2 | Threshold conceptions and procedural thresholds

Perkins (2006) suggested that the difficulty of some concepts may not stem from the concepts per se, but rather from the way some concepts interact with each other to create an underlying game which causes a deep transformation on students' understanding. For example, Land et al. (2005) explain that in computer programming the concepts of class, objects, tables, arrays and recursion may not have the troublesome or transformative characteristic but what is troublesome and transformative for students is the way that these concepts fit together and interact "*in a process of ever-increasing complexity*" (Land, Cousin, Meyer & Davies, 2005, p. 56). These concepts were characterised as threshold conceptions for they bind together aspects of a subject that may seem quite disparate to a novice "*but are fundamental to ways of thinking and practising in that discipline*" (Land, Cousin, Meyer & Davies, 2005, p.54).

Procedural or modelling thresholds were described by Davies & Mangan (2007). In their work, they specifically identified three types of conceptual change: basic, discipline and procedural or modelling. They locate the threshold concepts in the discipline and procedural conceptual change and they define them as "*understanding of other subject discipline ideas integrated and trans- formed through acquisition of theoretical perspective*" and "*ability to construct discipline specific narratives and arguments transformed through acquisition of ways of practicing*" correspondingly (Davies & Mangan, 2007, p. 1). These procedural thresholds can be linked with threshold skills as proposed by Thomas, Boustedt, Eckerdal, McCartney, Moström, Sanders, & Zander (2017) who reflect on skills as a form of procedural knowledge "*difficult or impossible to write down and difficult to teach best taught by demonstration and best learned by practice*" (Norman 1990 cited in Thomas et al. 2017, p. 335) and they suggest five key features of a threshold skill: transformative, integrative, troublesome, semi-irreversible and associated with practice.

Indeed, in computer programming there is a distinction between the difficulties that stem from conceptual knowledge (understanding of programming concepts) and difficulties stemming from procedural knowledge (knowing how). Upon this issue, many frameworks were articulated as early as 1980s. For instance, Bayman and Mayer (1998) recognised three types of programming knowledge, namely, syntactic knowledge, conceptual knowledge and strategic knowledge. A well-articulated framework was presented by McGill and Volet (1997). In this framework, they highlighted the following five areas of knowledge in programming: declarative-syntactic, declarative-conceptual, procedural-syntactic, procedural-conceptual, strategic/conditional knowledge.

2.3 | Challenges in identifying threshold concepts

Identifying threshold concepts in a discipline entails many challenges. Two of the most critical challenges refer to the method that should be employed and the participants that need to be involved.

Regarding the method, researchers have usually employed qualitative methods (e.g., interviews); however, a concrete methodology has not yet been established as the most appropriate methodology for identifying threshold concepts. The most commonly used method includes interviews with students and/or academics in which the researchers ask questions regarding difficult and challenging parts of the curriculum and changes that may have occurred as a result of overcoming these obstacles. Whereas this approach is appropriate for the aims of such a research, when employed alone may result in controversial results as each participant may reflect on a different concept and thus the depth that this concept will be investigated is superficial. Therefore, identifying threshold concepts in a discipline is difficult, requires time, reflection and debate. For this reason, Barradell (2013) argues that at least in the beginning, when an initial list of potential thresholds needs to be collected, consensus among the participants is significant. She further suggests the use of the Nominal Group Technique (NGT) or the Delphi Technique, methods that are used for investigating the collective opinion of a group of participants. Drawing on Barradell's suggestion, our first study conducted in this area employed the Delphi method and resulted in an initial list of 11 potential threshold concepts in the area of functions in programming. In this study, we proceed with interviews but focusing on a specific set of concepts to investigate this phenomenon deeper.

Another challenge in this research area refers to the participants and who is regarded as the most appropriate to reflect on his/her experiences to reveal transformations occurred when learning obstacles were overcome. Male and Baillie (2011:252) argue that to identify threshold concepts the most appropriate source is to collect data either directly from students or from "people whose experiences give them awareness of students' experiences". They further argue that teachers can identify not only concepts that are troublesome but concepts that are transformative for students. Additionally, Shinners and Kennedy (2013) point out that research in this field has reached a dead end. They criticise the methods used for identifying threshold concepts in programming, emphasising that asking students about difficulties they confronted in the past is an unreliable method. Instead, they advocate interviewing teachers to identify threshold concepts. They note that "there, after all, is where the reality of student learning is lodged, in the day-to-day classroom experience" (2013:14). They also state that the identification of threshold concepts needs both pedagogical and content knowledge on behalf of the interviewees and they suggest that new research attempts should direct their focus on teachers' pedagogical content knowledge to create teachers' concept representations (CoRe's). For this reason, Zwaneveld et al. (2016) contend that this can be found in secondary teachers rather than university teachers. They further advocate an approach that employs both teachers and students for this process. Based on this criticism, in the current research, we selected as participants secondary computer science teachers with many years of teaching experience.

In conclusion, even though a substantial amount of research studies has been conducted to identify threshold concepts in various disciplines, the question yet remains on how researchers should approach the identification problem.

3 | METHODOLOGY

Exploring if a concept is a threshold to students' progress, requires the identification of evidence in students' and teachers' experiences that would mirror the threshold concept characteristics and particularly the troublesome and integrative aspect of a concept, and the transformations that occur on the learners' understandings and/or identity once the concept is grasped. To this end, research in this area calls for qualitative approaches that are suitable for uncovering the participants' experiences, perspectives and meaning making of the phenomenon.

Interpretative Phenomenological Analysis (IPA), according to Charlick, Pincombe, McKellar & Fielder (2016), is based on both the descriptive and interpretive hermeneutic phenomenology. IPA draws from three methodological areas: phenomenology, hermeneutics and idiography (Larkin & Thompson, 2011). This approach is not common in computer science education studies although researchers in the more general field of computers and their use have employed this method to explain the participants' experiences (e.g., Symeonides & Childs, 2015). We decided that IPA was appropriate as a method of analysis for exploring the teachers' experiences with programming concepts in depth and the meaning they allocate to these experiences. IPA enabled us to conduct a thorough investigation and to offer an interpretative account of the teachers' meaning-making of their experiences with threshold concepts during their practice. In this study we summarise teachers' experiences that are of interest.

Our decision to focus on teachers' experiences as the primary emphasis of our investigation is based on an extensive literature review on threshold concepts and on other researchers' arguments and suggestions. Aligned with these researchers, we suggest that asking secondary students about the ontological and epistemological changes that they experience once specific concepts were understood may be an extremely difficult and unreliable endeavor and therefore, our attention on this phase of our research has focused on teachers whose experiences teaching these concepts can give us awareness of students' experiences. It is the teacher's job, through teaching, to observe and interpret and try to understand the difficulties that students' encounter in the course as well as to understand when the students finally surpass their problems by observing changes in their attitudes, emotions, behaviors and performance. As Allison & Pissanos (1994, p.47] argue "*observing hold a key position in the cycle (observing, interpreting, decision making) because interpretation of classroom events and, consequently, pedagogical decisions are dependent on the observational abilities of the teacher*."

3.1 | Participants

The participants selected for this study represent a homogeneous purposive sample. IPA necessitates a homogenous group of individuals which denotes that the participants should demonstrate experience with the same phenomenon (Creswell, 2007). In IPA, the researcher analyses the similarities or differences in a homogenous group, a group that is regarded similar regarding some characteristics (Pietkiewicz & Smith, 2014). For this reason, purposeful sampling or criterion-based selection is usually employed, giving the researcher a responsibility to select participants who have an important and meaningful experience of the phenomenon (Yuksel & Yildirim, 2015). In consideration of these issues, we chose purposive sampling to identify the participants. Thus, we selected the sample based on the purpose of the research, using teachers who had many years of experience teaching computer programming at key stages 4 and 5 (year 9–12 in the U.S. grade system).

To advertise the study, we created a call explaining the criteria for the participants and e-mailed it to UK master teachers and other computing teachers' networks such as CAS London. The criteria we listed were the following: the teachers should have experienced teaching programming at key stage 4 and 5 (lower and upper

TABLE 1 Teachers' characteristics

	Years of teaching experience in secondary settings	Years of practicing programming in professional settings	CAS master teachers
Rea	>7 years	<4 years	No
Olivia	>7 years	>4 years	Yes
Andrea	>7 years	<4 years	No
Mateo	>7 years	>4 years	Yes

secondary education) with more than 5 years of teaching experience. Ideally, we also asked for the participants to have some experience in practising programming at a professional level (Table 1 summarises the participants' characteristics). This call was part of our first study (Kallia & Sentance, 2017) in which ten computer science teachers took part. For this study, we specifically asked these ten teachers if they would like to participate in a follow-up interview. Including teachers that had also participated in the previous phase of our study was significant because we wanted to explore the experience of the teachers that had suggested the concepts under investigation as potential threshold concepts. The goal of this research was not to generalise these experiences to a population as this is the focus of quantitative research. The goal was to select participants whose experiences would give us rich insights into this phenomenon, and would improve its understanding. We argue that threshold concepts is a multifaceted phenomenon and would be experienced and come to light in different ways by different participants. However, these different experiences taken together would reveal aspects and characteristics of these concepts that constitutes them thresholds to the learner's progress.

Because IPA focuses on a thorough case exploration, a recommended sample is a small number of individuals who will enable a detailed examination of each case. As generalisation is not the purpose of the IPA studies, large samples are not advised; on the contrary, due to the idiographic characteristic of IPA, small samples are suggested to prevent the loss of important meaning (Brocki & Wearden, 2006). As such, we determined the number of participants for this phase by considering Pietkiewicz & Smith (2014) criteria: "(a) the depth of analysis of a single case, (b) the richness of the individual cases (c) how the researcher wants to compare and contrast single cases and (d) the pragmatic restrictions one is working under". In consideration of these criteria, and of Boyd (2001) point that a size of 2 to 10 participants are satisfactory to reach saturation, we initially aimed at a size of 4 participants. At this point, we checked the depth and richness of the interview data and, because we found it to be adequate, no more participants were recruited. As Pietkiewicz & Smith (2014) highlight, the researcher should focus on the depth of the interview data and not on the breadth of the sample.

Ethical approval was approved by the ethics committee of King's College London and the consent forms and information sheets were administered correspondingly. All participants were computing teachers with more than 7 years' teaching experience in programming at Key stages 4 and 5. To keep the teachers' participation anonymous we have changed their names in this study: teachers' names are Rea, Olivia, Andrea and Mateo. Specifically, Mateo and Olivia are Masters teachers[1] and they have worked as computing teachers for more than years. They have also some years of experience working outside school where they practiced programming for 1 to 3 years. Andrea and Rea have also taught computing in UK schools at key stage 4/5 for more than 7 years. Both have experience practicing programming for more than 4 years outside school settings.

3.2 | Data collection

The data was collected through semi-structured interviews which is the most common data collection method of IPA (Creswell, 2007). For the semi structured interviews, we developed a prompt sheet based on the concepts that resulted from the first and second phase of this study. These concepts were the basis of our conversation with the interviewees. Three of the teachers asked to be interviewed via videoconferencing and one teacher chose to visit us at our university.

The teachers' interviews began with the researcher explaining the research aim and purpose of the interview. After that, the participants were asked to discuss and reflect on their experiences teaching the concepts and how students are engaged with them. The interviews were recorded and lasted approximately 40 to 70 min and continued until the researcher felt that the participants could not share more things on the topic.

3.3 | Data analysis, reliability and trustworthiness

After the collection of the data, the interview records were transcribed. The data were analysed by employing IPA and following the guidelines suggested by Smith & Osborn (2003). The tool we used for the analysis of the interviews was Atlas.ti 7. The first author, following the IPA guidelines analysed each verbatim transcript independently. Because the focus of the analysis was to find evidence in the participants' experiences that corresponds to the threshold concept characteristics, the analysis paid extra attention on coding participants' experiences that demonstrate: (a) difficulties that students experience with the corresponding concept (the troublesome aspect of threshold concepts), (b) a deeper conceptual understanding of a concept and/or idea (the integrative part of threshold concepts) and (c) epistemological and ontological changes (the transformative aspect of threshold concepts) occurring once a concept is understood. Therefore, as the next section demonstrates (Section 4), the superordinate themes refer to difficulties, conceptual changes and coherence, and transformations for each concept examined. The analysis followed a mixed approach of deductive and inductive coding. The aforementioned pre-defined superordinate themes were used as a general guide for the coding process. However, a more in depth analysis (inductive approach) was followed to uncover themes related to how the participants experience the phenomenon and to explore differences or commonalities in the experiences of the participants.

Consequently, each case was analysed separately and emergent themes were created with the hermeneutic cycle being an important part of the analysis. The next step was to search for connections between the themes and to group them into clusters with each one of them containing themes with some conceptual similarities (Shinebourne, 2011). The outcome of this phase was a table with superordinate themes (addressing troublesome, integrative and transformative characteristics) and links to the lines of the transcript on which they can be located. This process was repeated for each case, and, thus, the author was vigilant to view each case separately and bracket the outcomes of the previous ones. As soon as the analysis was completed for all the cases separately, a final table was constructed—a consolidated list—including themes for the study as a whole. This again is an iterative process which requires going back and forth in the transcripts to merge themes or reject themes depending on the richness of the data (Shinebourne, 2011). Two external researchers coded again 2 of the 4 interviews using the final table, and the total percentage of agreement was calculated in Atlas.ti (a = 68.1). Some of the disagreement between the researchers mostly referred to the concepts that we did not find much evidence supporting the argument of being threshold concepts (e.g., control flow). For that reason after a discussion between the researchers these concepts were disregarded from the research. The percentage of agreement between the coders was calculated again (a = 75.1). Regarding the concepts that are presented in this paper, any disagreement between the researchers was discussed and resolved whether by changing or adapting the themes and coding were necessary.

The final consolidated table provided all the information that we needed to generate the narrative of the study which includes an interchange between the individuals' account -the participants' own words- and our interpretative stance. By doing that, the narratives include the participants' voice while enabling the readers to evaluate the accuracy of our interpretations. In the generation of our interpretations, following Collins & Nicolson (2002) suggestions, we tried to minimise our bias by reading the transcripts many times to make sure that our interpretations are indeed based on the participants' account. For this reason, we based our interpretation on the criterion of "grounding in examples," giving in this way the opportunity to readers to evaluate our interpretations. Following Flowers, Duncan & Frankis (2000) recommendation, we tried to provide the most representative extracts for each of the themes produced. Additionally, both the external researchers that assist us with the coding validate our interpretations as well.

To increase the trustworthiness, we followed Lincoln's and Guba's (1986) criteria and suggested techniques. In particular, for credibility and confirmability, prolonged engagement and investigator triangulation was used as it was described in the above section. For dependability, the data collection and analysis method are described in detail as well as the intercoder reliability was calculated and presented above. Finally, for transferability, we tried to provide as much as possible a thick and detailed description of the data collection, the characteristics of the participants that took part, the place of the interviews and the questions-the interview protocol employed for our research which is presented in the appendix.

4 | RESULTS

This section includes the results from the study. For each theme, we provide only the most representative extracts.

4.1 | Teachers' experiences teaching parameters, parameter passing and return values

During the interviews, teachers were asked about parameters, parameter passing, and return values. Specifically, they were asked about these concepts' difficulties for students as well as the changes in students' understandings after they have grasped these concepts from their own subjective experience. Although the teachers were asked about their experiences separately for each of these concepts, it is quite interesting that they sometimes referred to these concepts as a group of concepts. This was particularly evident when they were asked about changes in students' understandings or their programming attitude in general.

4.1.1 | Parameter passing is difficult for students

Teachers explained the reason of parameter passing difficulty for students. Reflecting on her experience as a teacher, Rea captured these difficulties by saying:

> "I think it [parameter passing] is difficult for students … the difficulty lies on how these variables in the function definition are going to take the values from the arguments. They can call a function, but they don't fully understand of how arguments are being handled. The key concept is when you are calling a function, and you are passing certain parameters in your function call, how are they being held in the function definition. That's the key thing that students struggle to understand." [Rea]

In the first sentence, Rea emphasises the relationship between arguments and parameters and the conceptual difficulty that this relationship imposes on students. Rea underlines the conceptual and not the procedural difficulty as she clearly demonstrates that students can call a function correctly and pass the corresponding arguments but how these arguments are being handled is still something that students struggle to understand. Thus, for Rea, the difficulty with understanding parameter passing focuses around students' endeavour to comprehend how the arguments in the calling statement are transferred in the function definition and are stored in the corresponding parameters.

Another problem was also mentioned by Rea. She explained that her students have difficulties when they call a function with more than one argument. She reports that her students often make syntactical errors in the calling statement, for example, omission of comma between the arguments or omission of parentheses:

> "When the function is called with a single argument or multiple, that's something that they don't fully understand … They don't separate the arguments with the comma, and then a classic mistake is that when they call a function they just forget to use the brackets. It's not that they have not understood it, it's like they are rushing through it. These are the normal syntax errors they make." [Rea]

Drawing on her teaching experience, Olivia emphasised that students are confused when they have to pass variables as arguments because this requires a more abstract way of thinking than passing simple values. She demonstrates this by saying:

> "… but I think if they call a subroutine and they put in the value that's going to be used, … they understand that much better than if they pass in a variable. Because that's more abstract, isn't it? That takes a bigger understanding of the whole program." [Olivia]

Olivia also reflects on another source of difficulty for students. She describes that there is a leap students have to make to understand how parameter passing works:

> "I think it's the leap from a basic program. They can understand a set of instructions X, Y, Z that's executed in that order. The difference between sort of instructions that are out of line … they'd understand it better if they had a real-life application for it. If you just try and explain to them in the programming language it's hard for them to grasp but if you begin to explain to them with a sort of real-life example they begin to understand it a bit better. Part of it is language also and what doesn't help with that is that different examples have different vocabulary and even that changes." [Olivia]

From Olivia's extract it is obvious that the teacher refers to the flow of the program and to the understanding that in programming some lines of code are executed sequentially while others are not. According to her experiences, students find it hard to make this transition. It would be helpful, she argues, if teachers used real-life examples to demonstrate this. Finally, she places the role of computing language among the main factors that may prohibit students from understanding this concept.

4.1.2 | Parameters are conceptually difficult for students

All four teachers agreed that students confront many difficulties in understanding the concept of a parameter. When Andrea was asked about what is difficult with understanding parameters she replied:

> "With parameters … when we are looking at how you can define a function, they don't understand … they think where these variables come from? Is it something that I'm already using in my program or not? They don't quite get that idea even though they use predefined functions and they pass their arguments into predefined functions. They cannot link what they're doing with the functions that they are in Python to the functions that they're doing themselves. So that analogy it doesn't work at the beginning." [Andrea]

Here, Andrea explains that students experience problems with understanding how to decide and define the corresponding parameters for the functions they create. She notes that even if students can use correctly predefined functions and pass the arguments appropriately, they seem not to be able to do the same with their functions. What the teacher tries to say here, is perhaps that in predefined functions the students only deal with concrete things like what arguments to pass to the calling function and do not have to think about more abstract notions as the parameter.

Yet, for Olivia, another source of the problem seems to be the language. She explains that students, in general, have problems understanding the vocabulary of computing and this is a deterrent factor for the conceptual understanding of the subject:

> "I think part of the problem [with parameters] is the language. In order for them to understand they need the vocabulary first … that sometimes is the barrier. It's not the fundamental understanding of a parameter or an argument, it's the language that makes it sound too scary. So, if you strip that away or if you develop an understanding of the vocabulary it becomes much easier to understand. I think with a lot of these concepts that that's the big issue, the language." [Olivia]

The teacher's observation about the difficulties that the vocabulary imposes on students reveals the multifaceted difficulties that they encounter in programming. Indeed, the teacher explains that it is not just understanding of what a parameter is or an argument is, but students may, in the beginning be intimidated by the new vocabulary that they are introduced to. Being afraid to engage with something unfamiliar to their own discourse experiences may prohibit their learning.

4.1.3 | Return values are conceptually difficult for students

All the participants discussed the difficulty that return values cause to students. Looking at an extract from Rea, it is obvious that the teacher focuses more on how students are handling the function's return value in the main function (or in the point that it is returned to). She explains that students have difficulties in understanding why the return value can be stored in a variable. Rea also explains that students' way of handling this demonstrates if they have understood return values or not. For example, if students use a print statement to print the function's outcome instead of using a variable first to store the outcome, this is evidence that students have not yet completely grasped the notion of return values.

> "If they get the concept of the function call and arguments and everything, what is difficult for them to understand is when you return something that needs to be caught in something. They are using a variable to catch it usually in where they call the function. Sometimes they just send it to a print function and if they do that they don't fully understand it, so I try to enforce the concept that catch it into a variable, so when a function returns a value that is caught into a variable then you can print that variable to see what is the return value." [Rea]

For Andrea, students' difficulties lie on a deeper level of understanding. She maintains that students fail to see the reason why a function should return something back and even though when they use predefined functions, it seems that they have captured this idea, when they turn to their functions they are still confused with this concept:

> "Sometimes they don't understand in the initial stages why something needs to be returned. Because normally they're used to working with just one program … even so they use functions like for example random or something like this and they do return values and they use these values, when they start creating their own they still get confused." [Andrea]

Olivia concentrates more on the teaching approach used to explain this concept to students. She advocates the use of real examples, like a factory, to demonstrate the need and use of returning something back. On the contrary, she explains that if teachers are too abstract from the beginning, this will encumber students' confusion:

> "I think it depends on how you teach it [return values]. I think if you are too abstract too quickly they panic a bit and therefore that panic stops them from seeing the bigger picture. But if you get them to role model, for example a factory, where you do a particular job and then you pass that finished part of that job back to the main bit of the factory, they understand that. And I think it's that development … from concrete to abstract … that they get to the point where they can do it themselves, I think that's the important part." [Olivia]

4.1.4 | Conceptual change and coherence

The teachers also reflected on their experiences on changes occurring in students on a conceptual level once they grasp these concepts. Specifically, reflecting on their experience, they were asked the following question: "*Is there another concept or something else that students understand in programming as soon as they grasp parameter passing*?" Rea highlighted the issues around variables, parameters, arguments and the flow of the program once parameter passing is grasped:

> "I think they understand that a variable is like a container. That's the key concept that they get straight away. It's the integration of what they have learnt previously and then they can develop a link of why it works. Parameters and arguments also take another conceptual shape as soon as they understand how parameter passing works. And, … they also understand that when you call a function what exactly happens. The flow of the program makes more sense to them… I can recall one of the students passing a comment: oh yeah now I get it. That was really pleasing. They are grasping a concept in more depth." [Rea]

There is much to analyse in this extract. Rea explains that parameter passing enhances students' understanding of variables. She refers to an integration of knowledge taking place where the students can connect what they already know with their new knowledge. Rea also notes that once students grasp how parameter passing works and what exactly is happening during this process, they start seeing the connection between parameters and arguments and how these communicate. At the end of the transcript, the teacher also explains that the flow of the program and apparently the non-sequential execution of the code starts making more sense to the students. For Rea, it may be that students begin to understand how data are transferred through the code and how different parts of the program communicate with each other.

Teachers were also asked about changes that occur on students once they understand parameters: "*Is there another concept or something else that students understand in programming as soon as they grasp parameters*?" Rea, Andrea and Olivia commonly agreed that students better realise the notion of variables or that the notion of variables is enhanced or further extended:

> "For example, in the lesson I taught them, I explained about parameters and variables and then they actually understood that in the function definition we can give it any name, like I can call it X,Y but when I am calling it I may be passing a value 5 and 6, so X goes 5 and Y goes 6, that's the whole concept of those variables, the parameters in the function are actually catching those values and then you can manipulate them as whatever

way you like in your call. So, yeah, that definitely consolidates the concept of what is a variable." [Rea]

The way the passage unfolds strongly suggests the strong connection that Rea makes between variables and parameters. She starts first by noting that it is important in teaching to highlight the connection between parameters and variables. She explains that once this is comprehensible by students, they can see that parameters can have any name, as variables do. In contrast to what they have been used to, the values of these variables are assigned to them from the calling statement, and specifically from the arguments. As such, the teacher's experience indicates that students can then understand that parameters can take different values that could potentially change the outcome of the function. Looking at the following extract, Andrea seems to have been engaged in a similar experience to Rea with her students:

"… because now the notion of variable is extended because they can see this is not just something we use in the main program, it might have some different scopes and different roles depending on the context. So that [parameters' understanding] certainly improves the variable understanding." [Andrea]

Andrea's extract captures the transformation or the extension of students' knowledge about variables. As she explains, students can see that variable are not only used in the main function of a program and consequently on calculations, but they can also have other roles like that of a parameter or an argument.

When Andrea was asked about changes in students' understanding having grasped the concept of return values, she explained that students' understanding of calling a function is enhanced as well as the relationship between return values and what is going on in the main program. In other words, she suggests that students understand how return values are being used and handled in the main program:

"… so with return values I think maybe they understand better the call of the function so they can see how that affects what's happening in the main program, how it can be used, how to calculate or produce some results." [Andrea]

Table 2 summarises the teachers' experiences with students' difficulties with these concepts and the conceptual connections the students make with other relevant concepts in the field.

4.1.5 | Students' transformations

While all four teachers demonstrate the transformation on students once these concepts are understood, there are similarities but also differences in how this transformation is portrayed. Most of the teachers think that these concepts' understandings and the interaction among them are essential to students' competence in programming and in understanding the role of functions and what they can do with them in their programs.

"It's difficult for me to explain. Once they understand all these, parameter passing, returns, parameters, arguments then it's something like clicked into their minds and understand what's going on." [Rea]

Looking at Rea's extract, it is obvious the importance that the teacher gives to these concepts with regard to students' understandings of functions. She explains that it is not clear what exactly is going on in the students' minds that makes them understand things that previously were not clear to them.

However, she notes that it is like something clicked into students' minds and they understand better what is happening.

Andrea aptly describes the students' personal transformation by saying:

"I think with parameters and parameter passing they can look at practical problems in a slightly different way and they probably even can think of like real-life scenarios. So … some analogies are quite helpful for a lot of children … it's important to understand that computation is not just something that happens with a

TABLE 2 Summary of teachers' experiences with students' difficulties and conceptual changes

	Teachers' experiences with difficulties students encounter	Conceptual connections
Parameter passing	a. How and from where parameters take their values /how arguments are being handled b. Multiple arguments are difficult for students to handle c. Syntax errors d. Passing arguments as variables is conceptually difficult e. How does the flow of the program jump from the calling Statement to the function definition?	a. Variables b. Parameters c. Flow of the Program
Parameters	a. Difficult to decide the parameters that a function needs b. Language too abstract	a. Variables
Return values	a. Where do return values go and how are they handled? b. Why are return values needed? c. Teaching methods too abstract	a. Calling a Function b. Data flow

computer. And I think [now] they'll look at other problems that involve information processing." [Andrea]

The way the extract unfolds strongly suggests that Andrea has experienced a change in her students' way of thinking as a result of understanding parameters and parameter passing. This is reflected from the beginning of the transcript where she describes the way that students are thinking about practical problems in a different way, while also considering real-life problems. The teacher's experience suggests that students start thinking about how programming is applied to everyday problems. Her experience has also led her to believe that her students see that computation is not something that happens from the computer itself but is something that starts first by their thinking processes which are then "transferred" or expressed in a way that computers can process. This is indeed a very strong transformative experience that Andrea demonstrates in this transcript.

Andrea also explains that as a result of fully comprehending parameters and parameter passing, the students' practical work becomes better. They also understand when there is a need to create a function while also can correct their errors by themselves. Both of these demonstrate a deeper level of understanding:

"I think in a lot of people, maybe the first change that you see is their practical work coming out better … because they understand a bit more and how to use it. And when they write their other functions and their new tasks, then they can correct the errors. They can understand how they're progressing better. I think practical change will be probably first and then if their practical work is successful then they can explain it better as well verbally or in writing. I think it generally improves their higher order skills. They can use abstraction in maybe more relevant, more efficient ways like they're looking like at the real-life problem. They can identify what parameters are needed for a particular subroutine or how it's going to help to make the solution more effective and efficient. So, I think that's all, how they need to decompose the problem using various functions." [Andrea]

The teacher, further reflecting on her experience, continues by saying that by understanding these concepts students' higher order skills are enhanced, they can think more abstractly and also start considering the effectiveness and efficiency of their programs. Taking all this together, Andrea's experience teaching these concepts demonstrate that her students can locate and correct errors in programming, to think abstractly, to seek more effective and efficient solutions and to connect their experience in class with real-life problems. Surely, all these transformations reveal a change—an epistemological change—on students' way of seeing programming and themselves in this course.

Almost the same arguments were mentioned by the other teachers as well. For instance, Olivia talks about an improvement in students' skills and specifically in decomposing a problem and in understanding that the individual pieces of code can be reused which makes their programs more flexible. She also argues, as Andrea did, that students' understanding of the efficiency of code is enhanced. For her, this is the moment when students' confidence is increased and start developing the flair for programming. She argues that this is something that cannot be taught, but that nevertheless is still an important moment in the learning process:

"When they understand the whole thing about parameters, return statements and parameter passing, they can see that they can decompose a problem down and they can write a small piece of code that they can reuse because it is very flexible. I think that's when they really get their idea of an elegant piece of code, an efficient piece of code. I think that's where the flair comes in … you know that flair that you can't really teach … and definitely boosts their confidence. I think once they've mastered it it's almost like an intermediate point they've reached. And I think they get an enjoyment from it at that point also which … in itself gives a bit of momentum to their understanding and their learning." [Olivia]

Olivia also highlights that once students' understanding is completed with return values, students finally understand what is happening with the data across their programs. She suggests that students can understand how the data are being transferred from one point to another in their code. She also notes that it is important in this endeavour to explain the reason for using the right identifiers so as to be easier for students to understand and track what is happening in their code:

"I think it's [change when they understand return values] the understanding of what's happening to the data, … Looking at things jumping around and a key to that … is teach them about good identifiers really early on because if you have a good identifier …, you can almost read it in English if that makes sense." [Olivia]

For Mateo, students' competence and confidence in programming is increased once all these concepts are understood. He explains that this is depicted in the students' program's complexity while he argues that students start to understand the power that their programs can have that wasn't previously achievable without all these concepts. He also believes that this makes students start enjoying programming while engaging with it:

"… understanding all these concepts, make them feel more confident and their functions can be more or less complex, and students enjoy that … I think it can give their program greater power, so the program can then begin to do things like calculating scores or validating names that it couldn't do before." [Mateo]

TABLE 3 Summary of teachers' experience on changes in students once these concepts are grasped

Teachers' experiences seeing their students being transformed when parameters, parameter passing and return values are understood
a. Functions and their role in programming are better understood
b. Changes in the way of thinking about practical problems and real-life scenarios
c. Information processing
d. Practical work becomes better
e. Enhanced programming vocabulary and better way of explaining phenomena
f. Effective and efficient solutions become part of their thinking
g. Decomposition's value is appreciated
h. Higher order skills are improved
i. Abstract way of thinking is enhanced
j. Flexibility in their programs
k. Flair of programming is enhanced
l. Data flow is better understood
m. Confidence and competence are increased
n. Enjoy programming

TABLE 4 Teachers' focus on students' difficulties and transformations—parameter, parameter passing and return values

	Difficulty emphasis	Transformative emphasis
Rea	Conceptual and practical difficulties	Conceptual and students' practice
Olivia	Conceptual and practical difficulties	Conceptual, students' practice, students' flair of programming, students' confidence
Andrea	Conceptual difficulties	Conceptual, students' practice, students' skills, students' realisation programming's role to everyday problems
Mateo	Practical difficulties	Students' practice, students' confidence

Table 3 summarises teachers' experiences seeing their students' being transformed when the concepts of parameters, parameter passing and return values together are understood, while Table 4 provides the focus of teachers' experience on students' difficulties and transformations.

4.2 | Procedural decomposition

Another concept that the teachers discussed most frequently during the interview was the concept of procedural decomposition. The teachers expressed very interesting opinions for this concept which were focused on two things: firstly, whether they believe that the difficulty of this concept lies on its conceptual understanding or a skill that lies behind the concept; secondly, the changes on students' understandings or attitudes towards programming once the concept is grasped.

4.2.1 | Conceptual and practical difficulties

When teachers were asked about this concept and specifically if, from their experience, its difficulty stems from its conceptual understanding or a skill that students need to master, all teachers argued that students first needed to understand the concept in a more theoretical way and then needed a lot of practice to be able to use and master procedural decomposition. This is most powerfully captured in Andrea's extract:

> "It probably goes in parallel because their practice will inform their understanding and they probably cannot practice without at least a little bit of knowledge given to them. So, they need to grow side by side as they practice more maybe knowledge is provided and practiced again, and again, and again on a different level. With decomposition, I think it's possible to understand it but not really being able to use it." [Andrea]

Following Andrea's thinking, it clearly demonstrates the skill that is needed to capture this concept. Andrea, at the beginning of the text, explains that it is, of course, natural that students need first to be theoretically introduced to this concept before they start practising it. However, later in the extract, she demonstrates the role and importance of practice and particularly on different levels of difficulty. The most interesting point of her discussion is the last sentence: in this, she plainly says that even if students understand the role of decomposition and perhaps the benefits and why it is used in programming, this does not mean that they are able to employ decomposition in their programs. This is a clear indication of a skill behind this concept which obstructs students' learning and ability to break down a problem.

A similar experience is shown by the other teachers as well. Mateo and Olivia explain that students need first to understand why decomposition is important and then they need to practice reinforcing what they learn in order to understand its value:

> "I think it's a concept that needs to be understood and then it needs to be reinforced by practice. So, they've got to understand first of all why it's important. I think the practice reinforces early knowledge of why it's important." [Mateo] and "I think they need to practice it to see the value in decomposing something." [Olivia]

For Rea too, students need both the conceptual understanding and the practice to fully grasp the concept.

> "It's first a concept but you can't have the concept and not the practice, so it needs to be like a combination of both I suppose." [Rea]

While all participants referred to the difficulties students encounter with procedural decomposition, there are differences in how these are portrayed, but all highlight the skill rather than the conceptual part of the concept.

> "I think it's as all of the high-order thinking, it's really hard because it might work okay on a very simple problem, more artificial style of problem, but if it was a real-life scenario … then without any help they will be really confused as to how to apply the skills that they have learned on a simple problem to this one." [Andrea]

Andrea's extract demonstrates the teacher's experience with procedural decomposition and the difficulties her students encounter. She explains that decomposition needs higher order skills and, thus, her students find difficult to practice it when they face a real-life scenario. In these examples, the teacher argues that students cannot proceed without her help. Olivia seems engaged in a similar experience with her students:

> "I think it is quite tricky for them at the beginning. What's sometimes tricky is … that you can't give them a meaningful example at the beginning for them to code because that's beyond them. So, part of the balance is between giving them a low enough example, concrete enough example so that they can understand the programming concept while trying to keep them motivated enough on the big picture that eventually this will be worth it if that makes sense. It's not until they actually do a meaty piece of work that they actually do value the idea of breaking things down." [Olivia]

Olivia explains that in the beginning students find it quite difficult to practice decomposition. One of the reasons she mentions is that they cannot practice in a meaningful example as this is quite difficult for them. Olivia's experience reveals the challenge the teachers face to provide a low-difficulty example that can demonstrate the value of decomposition so that students can really understand its importance and how it is practiced. However, she explains that students need a substantial amount of time practicing to actually understand decomposition.

4.2.2 | The value of decomposition and students' flair for programming

When teachers were asked about the changes in students' understanding or the ways students see the discipline, or students' personal skills and attitudes towards programming, three teachers reflected on their experiences and provided strong transformational evidence of this concept.

> "I think once they understand it, the purpose of it, and when they've seen some examples, they understand the value of it … how it can be used in the task. But … it requires a lot of practice to see how it actually works and how it helps and how it affects the practical side of work as well. This kind of area helps people to really move on in their understanding and I think that decomposition is on a different level of thinking because it's a more general type of skill … " [Andrea]

Here, Andrea illustrates a change in students' ways of seeing decomposition. She considers that students start valuing its importance, and how they can use it in their programs to increase the effectiveness of their practical work. For Andrea, once decomposition is grasped, students' understanding is progressed to a different level of thinking.

> "I think that's where their flair comes in. There's almost a competitiveness if you like to get things to work as efficiently as possible. I think it's a boost for their ego." [Olivia]

The effect of Olivia's passage vividly demonstrates the central point of this theme. The teacher explains that learning and understanding decomposition leads to students developing the flair for programming. Olivia also considers that students learn to work and look for efficiency in their programs, but also it makes them feel better about themselves and raises their morale in programming. Mateo also explains that students can see what they can accomplish with decomposition. He says that students understand that once they break down a problem into parts, they can work at each of this part at a time. They do not necessarily need to work at each of the functions at the same time and tackle everything at once. Instead, they can go back and forth and refine or redesign their functions as they move along. At the end of the extract, Mateo highlights that once students really understand procedural decomposition, then functions and the control flow of the program are easier to manage:

> "… because once they've broken down what the steps and stages are, they can tackle each step at a time. They don't necessarily have to get each step completed in its entirety and perfectly because they can always go back to it but if they've broken down their functions, if their program is working in functions, even if each function isn't perfect …, you can go back and refine it, separately from the main program. You can't necessarily tackle everything at once. And I think that a clear understanding, I think a clear initial decomposition of the program will make the control flow and the functions easier to manage." [Mateo]

Mateo argues that students come to realise that it is uneconomic to repetitively use the same lines of code to accomplish the same thing and then understand the usefulness and effectiveness of using functions in their programs:

> "And they first of all they execute one line after another and it all gets very uneconomic and then one

TABLE 5 Teachers' experiences of procedural decomposition's difficulties and students' changes once it is understood

Difficulties from teachers' experiences	Transformations from teachers' experiences
a. Higher order thinking skills are required b. Real-life scenarios are difficult to be understood c. Demonstrating decomposition's value is difficult d. Lots of practice is needed to master it	a. Usefulness and effectiveness of decomposition in programming b. Value of decomposition in programming c. Practical work becomes better d. Their programming skills are expanded e. Flair of programming f. Efficiency in their programs g. Boost for their ego h. Multiple ways of addressing a problem

> can say … now can we do it in fewer lines? And we want to do it in fewer lines … because resources are finite. We want your program to run as fast as possible to make as little impact on the computer as possible so can we now compress any of these lines and then we can start using functions … I think that's probably the major difference and they're beginning to avoid the repetition." [Mateo]

Table 5 provides a summary of teachers' experiences of procedural decomposition difficulties and students' transformations once procedural decomposition is understood.

5 | DISCUSSION

In this section, we will discuss the findings presented previously, and answer the research questions relating to this study.

5.1 | What are computing teachers' experiences with respect to the teaching of parameters, parameter passing, return values and procedural decomposition?

During the interviews, teachers reflected on their experiences of teaching the concepts of parameters, parameter passing, return values and procedural decomposition. The four teachers, based on their experience, presented important aspects of these concepts and specifically the difficulties that students encounter with them as well as the changes that these concepts' understandings evoke on students' personal level, confidence, competence and the way they see programming or an aspect of it.

It has been suggested that the area of functions in computer programming incorporates many difficulties for students and many studies have referred to the problems that students experience with parameters, parameter passing and return values (Madison & Gifford, 1997; Fleury, 1991; Kallia & Sentance, 2019). Our findings echo previous studies that discuss students' difficulties with functions. For example, one of the studies that employs the same theoretical framework as we is the study of Miller, Settle & Lalor (2015). In this study, the authors argue that parameter passing is a threshold concept and they base their argument mostly on the difficulties that students encounter with this concept. They referred to students' difficulties with parameters and parameter passing such as problems with aligning parameters and syntactical errors when calling a function. Similarly, the teachers in our study reported alike experiences with their students. They highlighted students' problems with using more than one parameter and argument in the function call as well as syntactical errors and using variables as arguments instead of values.

Another problem our teachers reported about parameter passing was understanding how the flow of the program is affected by the function call and understanding how the data are transferred from the calling statement to the function definition. Interestingly, a similar problem was reported in Sleeman, Putnam, Baxter & Kospa (1984). They specifically identified two common errors: the first one refers to the statements inside a procedure which students regard that are executed in the order they appear and the second refers to the time in which procedures are executed which students erroneously think that this execution is happening when procedures are encountered in a top-to-bottom scan.

In the same context, Ragonis & Ben-Ari (2005) further reported students' difficulties in understanding where the values of the parameters come from and where the return value of a method is returned. Indeed, our teachers also reflected on the same problems with parameters and return values. They report that students and it difficult to understand how and from where parameters take their values and where the return value goes and how it is handled. The teachers also mention a confusion between the return statement and the print function which is a problem that Miller, Settle & Lalor (2015) also identified.

While our research did not offer any new information regarding students' difficulties in functions, it is the first study which provides empirical data regarding teacher experiences on changes the students experience once these concepts are understood. These changes refer to conceptual transformations and integration of knowledge as well as students' personal attitudes towards programming. All teachers reflected on their experiences seeing students being transformed once they understood these concepts. In the following paragraphs, we discuss these transformations along with our suppositions regarding their place in the threshold concept framework.

5.2 | Is there evidence that supports the nomination of these concepts as threshold concepts, skills or conceptions?

While evidence of transformation and integration of knowledge was evident in each of the concepts of parameter, parameter passing and return values, we are reluctant to suggest that each of these concepts are threshold concepts as this evidence were not robust enough.

However, when these concepts are grouped together the transformations reported are much stronger and, thus, we can argue that they are likely to form threshold conceptions as defined by Perkins (2006). In the following section, these transformations are discussed along with procedural decomposition.

5.2.1 | Parameters, parameter passing and return values

As presented in the results section, all the concepts taken separately parameters, parameter passing, return values, appear to have some transformative and integrative features. However, these concepts evoke stronger transformations when they create a group that includes parameters, parameter passing and return values together. Perkins (2006) refers to such a group as threshold conceptions. Specifically, he suggests that the difficulty and transformative feature of some concepts may not stem from the concepts per se rather from the way some concepts interact with each other to create an underlying game which causes a deep transformation on students' understanding.

The teachers first mentioned that once students grasp these concepts, their understanding of functions, their role, how and why are being used in programming become clearer. Students finally understand how the data in their program are being transferred and, thus, how different parts can communicate with each other. As a result, apart from the expected increased quality of the students' work, they also start considering real-life problems, scenarios and how information processing and programming can be used to handle these problems.

The code's effectiveness and efficiency become part of their thinking and their programs obtain greater power and complexity. Students see and understand the effectiveness of decomposing a problem and reusing the same code more abstractly. This way of thinking boost students' higher order skills and abstract way of thinking, characteristics that describe computer programmers. This is the moment that actually students advance their skill in programming as one of the teachers aptly noted. Taking everything into consideration, teachers' experiences suggest that overcoming the conceptual and practical difficulties of parameters, parameter passing and return values engage students in a transformational journey where in the end students' understandings and knowledge are enhanced and integrated, and their way of thinking reflects the one of the practitioners in this field. Therefore, it can be argued that students experience an epistemological and an ontological transformation as they try to think as computer programmers which lead us to the suggestion that these concepts together likely form a threshold conception.

5.2.2 | Procedural decomposition

Procedural decomposition was the most controversial concept of all the eleven concepts identified in our previous study. This is because teachers in that study suggested that this is more a skill that needs to be practiced rather than a concept that needs to be theoretically understood. Particularly in computer programming, there are skills that students need to practice to understand programming entirely and to be able to write code in an advanced level. For example, many studies in computer programming distinguish between declarative knowledge, which refers to knowledge about concepts and principles, and procedural knowledge which refers to the active usage of declarative knowledge to solve a problem (Palumbo, 1990; Lau & Yuen, 2009).

From the teachers' interviews around this concept, the evidence found suggests that procedural decomposition is a procedural threshold or a threshold skill rather than a threshold concept. This is because the teachers, even though they highlighted the theoretical knowledge that students need to understand first, emphasised that this is something that is reinforced by practice and students need a lot of practice to finally master decomposition. They also mentioned that even if students understand theoretically the role of procedural decomposition in programming, this will not imply that they can actually apply decomposition to their programs. This strongly suggests that there is a skill that needs to be practiced by students to fully grasp procedural decomposition.

Specifically, teachers' experience with teaching decomposition reveals that there is a transformation that students undergo once they master decomposition. The teachers argued that this transformation leads students to appreciate what they can actually achieve and accomplish by employing decomposition, and also valuing its role and importance in programming and to see how their programs can be more effectively and efficiently written. The teachers also mentioned that this is one of the moments where the flair for programming is developed and where the students start thinking about problems differently by applying higher order skills. The changes referred here reflect the kind of transformation that threshold skills provoke as defined by Thomas, Boustedt, Eckerdal et al. (2017, p.335) who argue that "*mastering a threshold skill transforms what students can do and their vision of what they can do*" while also enables students to see "*other possible applications, broadening the list of tasks student can perform or enabling them to perform them in a new way.*"

Taking everything into consideration, the changes the students experience, as reported by their teachers, once they master decomposition, indicate an epistemological shift where the students start developing their thinking in a way that echoes the scientific thinking of this field. Therefore, we suggest that procedural decomposition can possibly be seen as a procedural threshold (threshold skill) in computer programming.

5.3 | Impact on teaching computer programming and future research directions

The findings of our research could have a significant impact on the teaching of computer programming in secondary education but also in higher education. Specifically, the findings of this study affect three major education areas, curriculum design, teaching methods and instructional tools and are discussed in the following paragraphs.

In this study, we identified some elements of integrated knowledge in functions. Kinchin (2010) argues that dedicating time as educators to integrate prior knowledge is really important and the benefits may surpass the ones of attaining new knowledge that is likely to remain secluded and unrelated. Tobias (1994) considers prior knowledge as one of the most significant factors that impact learning and achievement. Teachers should be encouraged to take into consideration the powerful conceptual relationships and links that these concepts form and the transformations that students exhibit as a result of them. From this perspective, we argue that the programming curriculum should be designed "recursively," with opportunities for students to revisit knowledge, accommodate new understandings in existing schemas and extending these appropriately. The spiral or "recursive" programming instruction mirrors Bruner's (1960) spiral curriculum where ideas and concepts are initially introduced and mastered at a simpler or basic level and further revisited and reconstrued in a higher level. Therefore, one of our next research goals would take this problem into consideration and will explore ways to design a constructive, and spiral curriculum that helps students integrate previous with new knowledge effectively.

Additionally, the importance of transformations occurring during the learning process has long been documented by Mezirow (1991). The challenge in the current education system and for educators is on how to present new information after having previously revised or transformed students' existing knowledge. Therefore, having identified some of the students' transformations our next research goal will also identify appropriate teaching strategies and practices that will more easily bring about these transformations and assist students in their attempt to understand and identify themselves in the computer classroom. We are particularly interested in exploring transformative learning theory and different transformative learning and teaching strategies that could potentially develop a constructive teaching process appropriate for fostering transformative learning.

These teaching strategies should also consider the distinction between threshold concepts, conceptions and procedural thresholds. Although the literature thus far does not offer tangible guidelines on how teachers should approach thresholds, relevant research could be considered as a starting point of this investigation. For example, the debate of whether conceptual knowledge should proceed procedural knowledge or the other way around, it is pertinent in programming education as well as in the threshold concept framework. In mathematics education, for instance, Kadijevich (2018) points out that the focus should be on the relationships between these two forms rather than their precedence. On the same line, Rittle-Johnson and Alibali, (1999) argued that the relationship between conceptual and procedural is iterative meaning that advances in one lead to advances to the other and thus, an instruction should iterate between concepts and procedures. Thus, it would be interesting to investigate how this relationship and its direction develops inside the threshold concept framework and how threshold concepts or conceptions affect procedural thresholds and the other way around and the impact of the findings to teaching practice.

Finally, since computer programming in all levels of education is inextricably connected with computer programming tools, it will be worth investigating how our findings may affect the instructional strategies employed in parallel with these tools and their designing interface. The last years, there has been an increase of programming environments, both block-based and text-based; however, evaluations on how these tools help students build a concrete conceptual and procedural understanding and their interrelation in programming are still obscurely explored. It is critical, therefore, to consider and explore the design and pedagogical principles employed to build these tools as well as the way they are being employed in the classroom and their effect on students' understandings. Exploring this under the threshold concept framework would shed light on the way that various thresholds can be addressed with computer-assisted tools. For example, the literature suggests that visual tools can transform abstractions into tangible representations (Crews & Butterfield, 2002) and that flow charts and concept maps can help both teachers observe students' learning progress and students build a more concrete conceptual understanding in programming (e.g., Hubwieser & Mühling, 2011; dos Santos et al., 2017). This is particularly relevant to the integrative part of threshold concepts as concept maps as well as flow charts can depict relations between concepts and data respectively. It will be worth investigating how these visualization tools can be employed to address conceptual thresholds and threshold conceptions. Another area that seems promising for addressing students' understandings is block-based tools as research suggests that they support learners' conceptual understanding (Weintrop & Wilensky, 2016). The question that needs, however, to be addressed here is whether the block-based environments, under specific instructional strategies, can address students' misunderstandings, misconceptions and thereafter, thresholds to students' progress. Thus, future research should further investigate how instructional strategies and computer programming interfaces and tools can work together to develop a framework that can help students resolve thresholds and successfully pass through liminality.

6 | CONCLUSION

This study focused on teachers' experiences in functions and particularly their experiences teaching the concepts of parameters, parameter passing, return values and procedural decomposition. Based on the teachers' experience we first highlighted some common difficulties that students at key stages 4 and 5 encounter while trying to understand these concepts. This specific output is of great significance as it can be used by teachers who would like to organize their lessons around difficult points in this specific thematic area of the curriculum and prepare appropriate materials that will help students overcome the corresponding obstacles. Secondly, we explored students' transformation and integration of knowledge once these concepts are grasped. This is the first study that endeavours to investigate these transformations with the specific set of concepts and by employing an interpretative phenomenological analysis of interviews with experienced computing teachers. The results of this analysis led us to propose procedural decomposition as a possible procedural threshold (threshold skill) and the group of parameters, parameter passing and return values as a possible threshold conception. The paper summarises and discusses the findings as well as potential implications for the computer programming education field.

PEER REVIEW

The peer review history for this article is available at https://publons.com/publon/10.1111/jcal.12498.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ORCID

Maria Kallia https://orcid.org/0000-0002-8591-9651

ENDNOTE

[1] CAS Master Teachers "champion computer science in schools and the wider teaching profession, provide training, mentoring and coaching to teachers in their local communities, and support collaboration between schools and universities" (https://www.computingatschool.org.uk/custom_pages/36-master_teachers)

REFERENCES

Akerlind, G., McKenzie, J., & Lupton, M. (2010). A threshold concepts focus to first year law curriculum design: Supporting student learning using variation theory. Retrieved from http://fyhe.com.au/past_papers/papers10/content/pdf/12B.pdf. Paper presented at 13th Pacific Rim First Year in Higher Education Conference, Adelaide, Australia. Retrieved January 15, 2018.

Allison, P. C., & Pissanos, B. W. (1994). The teacher as observer. *Action in Teacher Education*, *15*, 47–54.

Barradell, S. (2013). The identification of threshold concepts: A review of theoretical complexities and methodological challenges. *Higher Education*, 65(2), 265-27, Springer

Bayman, P., & Mayer, R. (1998). Using conceptual model to teach BASIC computer programming. *Journal of Educational Psychology*, *80*(3), 291–298.

Blackie, M. A., Case, J. M., & Jawitz, J. (2010). Student-centredness: The link between transforming students and transforming ourselves. *Teaching in Higher Education*, *15*(6), 637–646.

Boustedt, J., Eckerdal, A., McCartney, R., Moström, J., Ratclifie, M., Sanders, K., & Zander, C. (2007). *Threshold concepts in computer science: Do they exist and are they useful? SIGCSE '07* (pp. 504–508). New York, NY: ACM. https://doi.org/10.1145/1227310.1227482

Boyd, C. (2001). Phenomenology the method. In P. L. Munhall (Ed.), *Nursing research: A qualitative perspective*. Sudbury, MA: Jones and Bartlett.

Brocki, J., & Wearden, A. (2006). A critical evaluation of the use of interpretative phenomenological analysis (ipa) in health psychology. *Psychology and Health*, *21*, 87–108.

Bruner, J.S. (1960) *The process of Education*. Harvard University Press. Cambridge.

Charlick, S., Pincombe, J., McKellar, L., & Fielder, A. (2016). Making sense of participant experiences: Interpretative phenomenological analysis in midwifery research. *International Journal of Doctoral Studies*, *11*, 205–216.

Clement, J., Lochhead, J., & Soloway, E. (1990). Positive effects of computer programming on students? understanding of variables and equations. In *ACM 1980 Proceedings of the ACM 1980 annual conference* (pp. 467–474). New York: ACM.

Collins, K., & Nicolson, P. (2002). The meaning of satisfaction for people with dermatological problems: Reassessing approaches to qualitative health psychology research. *Journal of Health Psychology*, *7*, 615–629.

Cook-Sather, A. (2014). Student-faculty partnership in explorations of pedagogical practice: A threshold concept in academic development. *International Journal for Academic Development*, *19*(3), 186–198.

Cousin, G. (2006). An introduction to threshold concepts. *Planet*, *17*, 4–5.

Creswell, J. (2007). *Qualitative inquiry and research design: Choosing among five approaches* (2nd ed.). Thousand Oaks, CA: Sage.

Crews, T., & Butterfield, J. (2002). Using technology to bring abstract concepts into focus: A programming case study. *Journal of Computing in Higher Education*, *13*(2), 25–50.

Davies, P., & Mangan, J. (2007). Threshold concepts and the integration of understanding in economics. *Studies in Higher Education*, *32*, 711–726.

Dos Santos, V., De Souza, É. F., Felizardo, K. R., & Vijaykumar, N. L. (2017). Analyzing the use of concept maps in computer science: A systematic mapping study. *Informatics in Education*, *16*(2), 257–288.

Eckerdal, A., McCartney, R., Moström, J., Ratcli_e, M., Sanders, K., & Zander, C. (2006). Putting threshold concepts into context in computer science education. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education ITiCSE '06* (pp. 103–107). Bologna, Italy: ACM. https://doi.org/10.1145/1140124.1140154

Eckerdal, A., McCartney, R., Moström, J., Sanders, K., Thomas, L., & Zander, C. (2007). From limen to lumen: Computing students in liminal spaces. In *Proceedings of the third international workshop on computing education research* (pp. 123–132). New York: ACM.

Eckerdal, A., & Thuné, M. (2005). Novice Java programmers' conceptions of "object" and "class", and variation theory. In *ITiCSE*, 27-29 June, Monte de Caparica, Portugal, New York: ACM.

Fleury, A. (1991). Parameter passing: The rules the students construct. In *Proceedings of the twenty-second SIGCSE technical symposium on computer science education* (pp. 283–286). New York: ACM.

Flowers, P., Duncan, B., & Frankis, J. (2000). Community, responsibility and culpability: Hiv risk-management amongst Scottish gay men. *Journal of Community and Applied Social Psychology*, *10*, 285–300.

Fox, R., & Farmer, M. (2011). The effect of computer programming education on the reasoning skills of high school students. In *Proceedings of the international conference on frontiers in education: Computer science and computer engineering* (pp. 187–193). Las Vegas, NV: FECS conference proceedings

Gourlay, L. (2009). Threshold practices: Becoming a student through academic literacies. *London Review of Education*, *7*(2), 181–192.

Holloway, M., Alpay, E., & Bull, A. (2010). A quantitative approach to identifying threshold concepts in engineering education. In *Engineering Education2010 (EE2010) Inspiring the next generation of engineers*. Loughborough, England: Higher Education Academy Engineering Subject Centre.

Hubwieser, P., & Mühling, A. (2011, August). What students (should) know about object oriented programming. In *Proceedings of the seventh international workshop on computing education research* (pp. 77–84). New York: ACM.

Kadijevich, D. M. (2018). Relating procedural and conceptual knowledge. *The Teaching of Mathematics*, *21*(1), 15–28.

Kallia, M., & Sentance, S. (2017). Computing teachers' perspectives on threshold concepts: Functions and procedural abstraction. In *Proceedings of the 12th workshop on primary and secondary computing education* (pp. 15–24). New York: ACM.

Kallia, M., & Sentance, S. (2019). Learning to use functions: The relationship between misconceptions and self-efficacy. *In Proceedings of the 50th ACM technical symposium on computer science education* (pp. 752-758). New York: ACM.

Kinchin, I. (2010). Solving cordelia's dilemma: Threshold concepts within a punctuated model of learning. *JBE*, *44*, 53–57.

Land, R., Cousin, G., Meyer, J., & Davies, P. (2005). Threshold concepts and troublesome knowledge (3): Implications for course design and evaluation. In Proceedings of the 12th Improving Student Learning Conference. In C. Rust (Ed.), *Improving Student Learning—Diversity and inclusivity* (pp. 53–64). Oxford: Oxford Centre for Staff and Learning Development.

Larkin, M., & Thompson, A. (2011). Interpretative phenomenological analysis. In A. Thompson & D. Harper (Eds.), *Qualitative research methods in mental health and psychotherapy: A guide for students and practitioners*. Oxford, England: John Wiley and Sons.

Lau, W., & Yuen, A. (2009). Exploring the effects of gender and learning styles on computer programming performance: Implications for programming pedagogy. *British Journal of Education Technology*, *40*, 696–712.

Lincoln, Y. S., & Guba, E. G. (1986). But is it rigorous? Trustworthiness and authenticity in naturalistic evaluation. *New Directions for Program Evaluation*, *1986*(30), 73–84.

Madison, S., & Gifford, J. (1997). Parameter passing: The conceptions novices construct. Retrieved from https://files.eric.ed.gov/fulltext/ED406211.pdf technical Report.

Male, S., & Baillie, C. (2011). Engineering threshold concepts. In *Proceedings of SEFI Annual Conference* (pp. 251–257). Brussels: SEFI, European Society for Engineering Education. Retrieved from http://www.sefi.be/wpcontent/papers2011/T7/24.pdf

McGill, T., Volet, S., & Hobbs, V. (1997). Studying programming externally: Who succeeds? *Distance Education*, *18*(2), 236–256.

Meyer, J., & Land, R. (2003). Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practicing. In C. Rust (Ed.), *Improving student learning-ten years on*. Oxford, England: OCSLD.

Mezirow, J. (1991). *Transformative dimensions of adult learning*. San Francisco, CA: Jossey-Bass.

Miller, C., Settle, A., & Lalor, J. (2015). Learning object-oriented programming in python: Toward an inventory of di_culties and testing pitfalls. In *Proceedings of the 16th annual conference on information technology education* (pp. 59–64). New York: ACM.

Norman, D. A. (1990). *The Design of Everyday Things*. Doubleday, New York

Palumbo, D. (1990). Programming language/problem-solving research: A review of relevant issues. *Review of Educational Research*, *60*, 65–89.

Pardamean, B., Honni, H., & Evelin, E. (2011). The effect of logo programming language for creativity and problem solving. In *Proceedings of the 10th WSEAS international conference on E-Activities* (pp. 151–156). Stevens Point, WI: World Scientific and Engineering Academy and Society.

Perkins, D. (1999). The many faces of constructivism. *Educational Leadership*, *57*, 6–11.

Perkins, D. (2006). The underlying game: Troublesome knowledge and threshold conceptions. In J. Meyer & R. Land (Eds.), *Overcoming barriers to student understanding: Threshold concepts and troublesome knowledge*, London: Routledge.

Pietkiewicz, I., & Smith, J. (2014). A practical guide to using interpretative phenomenological analysis in qualitative research psycholog. *Czasopismo Psychologiczne*, *20*, 7–14.

Popat, S., & Starkey, L. (2019). Learning to code or coding to learn? A systematic review. *Computers & Education*, *128*, 365–376.

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students? Reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science*, *45*, 583–602.

Ragonis, N., & Ben-Ari, M. (2005). A long-term investigation of the comprehension of oop concepts by novices. *Computer Science Education*, *15*, 203–221.

Rittle-Johnson, B., & Alibali, M. W. (1999). Conceptual and procedural knowledge of mathematics: Does one lead to the other? *Journal of Educational Psychology*, *91*(1), 175.

Rountree, J., & Rountree, N. (2009). Issues regarding threshold concepts in computer science. In *Proceedings of the Eleventh Australasian Conference on Computing Education—volume 95 ACE '09* (pp. 139–146). Wellington, New Zealand: Australian Computer Society, Inc. Retrieved from http://dl.acm.org/citation.cfm?id=1862712.1862733

Sandri, O. (2013). Threshold concepts, systems and learning for sustainability. *Environmental Education Research*, *19*, 810–822.

Shinebourne, P. (2011). The theoretical underpinnings of interpretative phenomenological analysis (ipa). *Journal of the Society for Existential Analysis*, *22*, 16–31.

Shinners-Kennedy, D., & Fincher, S. (2013). Identifying threshold concepts: From dead end to a new direction. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research. ICER '13* (pp. 9–18). San Diego, San California: ACM. https://doi.org/10.1145/2493394.2493396

Sleeman, D., Putnam, R., Baxter, J., & Kospa, L. (1984). Pascal and high school students: A study of misconceptions. Retrieved from https://files.eric.ed.gov/fulltext/ED258552.pdf research Report.

Smith, J., & Osborn, M. (2003). Interpretative phenomenological analysis. In J. A. Smith (Ed.), *Qualitative psychology: A practical guide to research methods*. London, England: Sage Publications.

Symeonides, R., & Childs, C. (2015). The personal experience of online learning: An interpretative phenomenological analysis. *Computers in Human Behavior*, *51*, 539–545.

Taylor, M., Harlow, A., & Forret, M. (2010). Using a computer programming environment and an interactive whiteboard to investigate some mathematical thinking. *Procedia Social and Behavioral Sciences*, *8*, 305–321.

Thomas, L., Boustedt, J., Eckerdal, A., McCartney, R., Moström, J., Sanders, K., & Zander, C. (2017). In the liminal space: Software design as a threshold skill. *Practice and Evidence of the Scholarship of Teaching and Learning in Higher Education*, *12*, 333–351.

Tobias, S. (1994). Interest, prior knowledge, and learning. *Review of Educational Research*, *64*, 37–54.

Tu, J., & Johnson, J. (1990). Can computer programming improve problem solving ability. In *ACM SIGCSE Bulletin* (pp. 30–33). New York: ACM.

Weintrop, D. and Wilensky, U., (2016). Bringing blocks-based programming into high school computer science classrooms. In Annual *Meeting of the American Educational Research Association (AERA)*. Washington, DC: AERA.

Wright, A., & Hibbert, P. (2015). Threshold concepts in theory and practice. *Journal of Management Education*, *39*, 443–451.

Yizhou, Q., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education*, *18*, 11–24. https://doi.org/10.1145/3077618

Yuksel, P., & Yildirim, S. (2015). Theoretical frameworks, methods, and procedures for conducting phenomenological studies in educational settings. *Turkish Online Journal of Qualitative Inquiry*, *6*, 1–20.

Zwaneveld, B., Perrenet, J., & Bloo, R. (2016). Discussion of methods for threshold research and an application in computer science. In R. Land, J. H. F. Meyer, & M. T. Flanagan (Eds.), *Threshold concepts in practice* (pp. 269–284). Rotterdam: Sense Publishers.

How to cite this article: Kallia M, Sentance S. Threshold concepts, conceptions and skills: Teachers' experiences with students' engagement in functions. *J Comput Assist Learn*. 2021;37:411–428. https://doi.org/10.1111/jcal.12498

APPENDIX A.

Basic interview questions

1. From your experience, what are the difficulties students face with parameters?
2. From your experience, what are the difficulties students face with parameter passing?
3. From your experience, what are the difficulties students face with return values?
4. From your experience, what are the difficulties students face with procedural decomposition?
5. Have you experienced any changes that happen to students once they grasped the concept of parameter passing? For example, is there another concept or something else that students understand in programming as soon as they grasp parameter passing?
6. Have you experienced any changes that happen to students once they grasped the concept of return values?
7. Have you experienced any changes that happen to students once they grasped the concept of parameters?
8. Have you experienced any changes that happen to students once they grasped the concept of procedural decomposition?
9. Based on your teaching experience, do you think that students have more difficulties in understanding the concept of procedural decomposition or applying it in programming exercises?