Introducing a framework for improving competitive programming education for the Cyprus Olympiad in Informatics

A project submitted to Middlesex University in partial fulfilment of the requirements for the degree of Doctorate in Professional Studies

Panayiotis Eracleous

Faculty of Professional and Social Sciences

January 2022

Disclaimer

The views expressed in this document are mine and are not necessarily the views of my supervisory team, examiners or Middlesex University.

Table of Contents

List of Figures

List of Tables

Glossary

ACM: Association for Computing Machinery

BFS: Breadth-First Search

BOI: Balkan Olympiad in Informatics

CCS: Cyprus Computer Society

CER: Computing Education Research

COI: Cyprus Olympiad in Informatics

CS: Computer Science

DFS: Depth-First Search

DP: Dynamic Programming

EJOI: European Junior Olympiad in Informatics

ICT: Information and Communications Technology

IOI: International Olympiad in Informatics

JBOI: Junior Balkan Olympiad in Informatics

KMP: Knuth Morris Pratt algorithm

LIFO: Last In First Out

MOEC: Ministry of Education and Culture of Cyprus

URL: Uniform Resource Locator

Abstract

The purpose of the study is to empower the next generation of Computer Science experts by helping high school students comprehend complex programming concepts and solve challenging tasks in programming competitions. The research aims to enhance the pedagogy and the teaching practice of competitive programming education by introducing and evaluating a framework as a training system and utilising a code-evaluation platform within the Cyprus Olympiad in Informatics (COI) (Eracleous et al., 2019). The proposed COI framework intends to redefine the teaching and learning processes within its discipline. The research focuses on four critical pillars: the pedagogical model of a learning community, the instructional strategy of Problem-Based Learning (PBL), mental models and strategies, and online technologies. In addition, the project introduces Michanicos, a code-evaluation platform that enables real-time code assessment and facilitates the application of the framework's scope in practice.

The research project reports on the empirical evaluation of the COI framework in the context of the COI course that selects and prepares the Cypriot delegations for international competitions. I used a constructivist approach with a combination of action research and mixed-methods assessment. The data was collected within a year from 125 participants using interviews, questionnaires and performance data. The research project has provided evidence that the COI framework is a reliable pedagogical method that supports students to increase their programming abilities and enables Cypriot delegations to improve their results in international competitions.

The research project provides three distinctive contributions to knowledge: (a) the identified threshold concepts of competitive programming, (b) the methodology for identifying threshold concepts, (c) the COI framework offers methods of inquiry to assess student performance in the liminal space by using programming tasks on Michanicos. The project has the potential to inform theory and practice in competitive programming education. Furthermore, it provides a method that can produce consistent results in international competitions by supporting educators and students in their preparation and threshold concepts researchers in their quest for unlocking new ways of thinking.

CHAPTER 1: Project Information

1.1 Introduction

Decades of research have proven that learning to programme is a complicated process for many students. As a result, programming courses have been associated with high student dropout rates (Bennedsen and Caspersen, 2007) and fragile learning (McCracken et al., 2001; McGettrick et al., 2005). This condition creates substantial challenges for programming educators, who want their students to progress and compete at the highest level. To investigate these challenges and understand student engagement with learning resources, I have explored the theory of threshold concepts (Meyer and Land, 2003) as a theoretical framework. Furthermore, I reviewed the theory-into-practice framework design by Dabbagh (2005) that suggests that educators need to have a reflexive awareness of the theoretical basis underlying instructional design.

For my research project, building on the work of Meyer and Land (2003) and Dabbagh (2005), I have developed a system of training for competitive programming, the COI framework, that I have integrated into the Cyprus Olympiad in Informatics programming course (Eracleous et al., 2019). Meyer and Land (2005) proposed using threshold concepts to distinguish specific concepts that can be used to organise the learning process with linkage to ways of thinking and practising. One of the COI framework's elements, the redesigned COI curriculum, includes identified threshold concepts that are complex for students but have potentially transformative effects. Meyer and Land (2005) introduce the liminal space, which explicitly focuses on teaching and learning these concepts. The COI framework supports the idea of releasing students into an intentionally created liminal space of uncertainty where I can identify students' strategies and assess performances with the identified threshold concepts.

Dabbagh (2005) proposes a theory-based design outline highlighting the transformative interactions between pedagogical models, teaching strategies and learning technologies. She argues that situated cognition is a practical knowledge perspective from which to develop pedagogical models for learning. Learning technologies can bring together a learning community and allow distributed forms of interaction and organisation of learning

activities to achieve a common goal. Moreover, she suggests that advances in web-based technologies have facilitated learning interactions and inspired the development of pedagogical models. These models are grounded in constructivist views to promote meaningful knowledge acquisition (Dabbagh, 2005).

The COI framework (Figure 1) capitalises on: the pedagogical model of a learning community, the teaching strategy of Problem-Based Learning, the learning technologies of a code-evaluation platform and a contest management system that direct the learning activities, and the worked examples of programmes that support the mental models' acquisition. I redesigned the COI curriculum around identified threshold concepts, and by investigating students' liminal spaces, I was able to target specific ways of thinking.

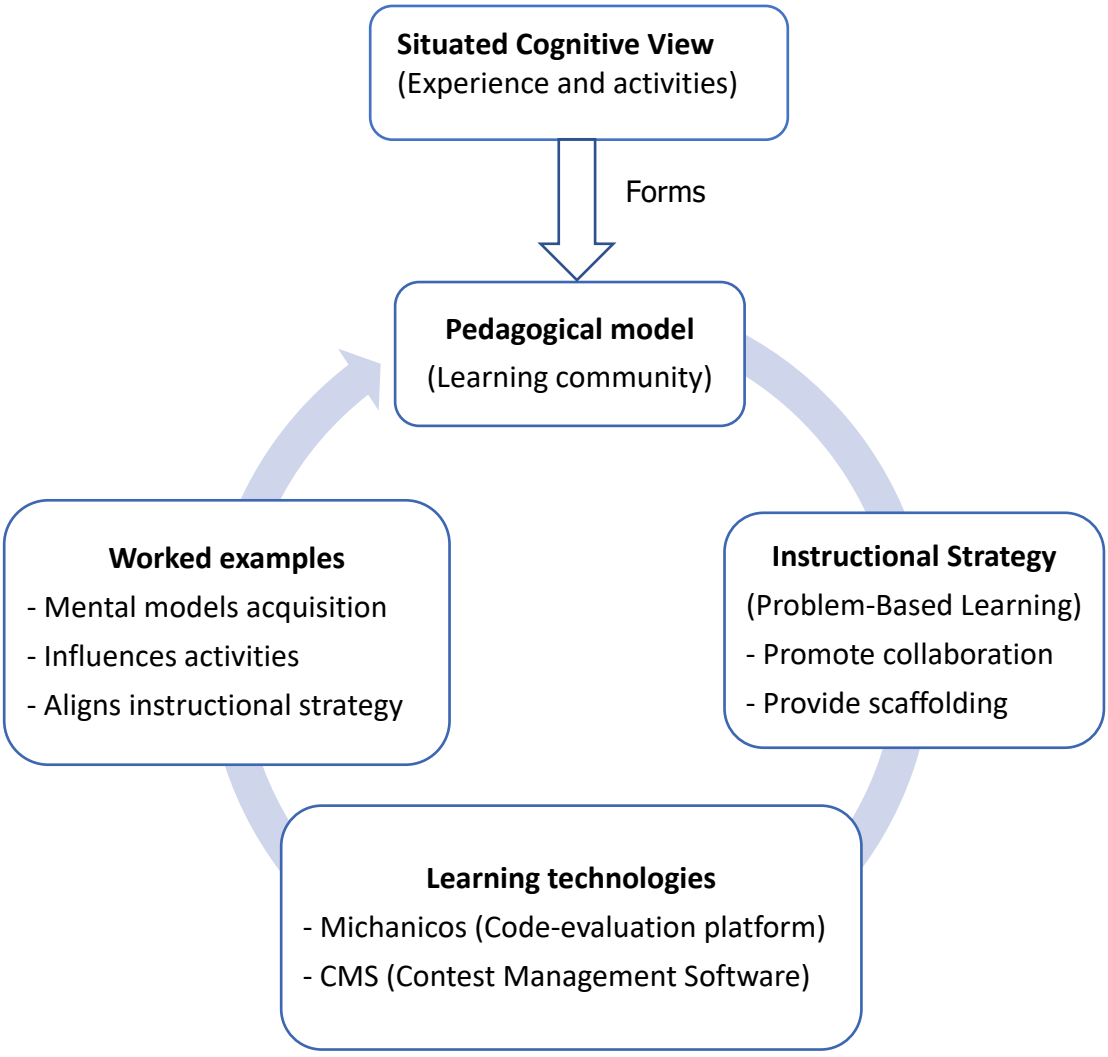


Figure 1: COI framework design

The constructivist theory holds that students actively build knowledge. The threshold concepts are the foundations of the structure that keep the remaining components together, and the liminal space is the construction site (Eckerdal et al., 2007). Advanced online technology can support the development of a constructivist learning environment, which is a robust structure for improving students' engagement, motivation, and learning skills (Yacob, 2012). In a constructivist environment, learning something new builds upon the students' prior knowledge and interests through their interaction with new experiences (Howe and Berv, 2000).

From a constructivist perspective, it is crucial to identify mental models (Johnson-Laird, 1983) to develop student learning. Mental models and, specifically, the notional machine (Du Boulay, 1986) are vital aspects of my framework, and their acquisition is supported by the provision of worked examples of programmes. The responsibility is put on the students to explore the liminal space, create the appropriate mental models, and the support is analogous to the struggles and rigour during the process of threshold capture. The Problem-Based Learning (PBL) approach employed by the COI framework as an instructional strategy matches this requirement. There is evidence that PBL, compared to other instructional methods, has significant value for enhancing student learning and problem-solving skills (Looi and Seyal, 2014).

Recognising the different levels of cognitive ability and programming experience, I propose a layered approach to scaffolding as the required support level for each student through three rounds of competitions may vary. Furthermore, the COI framework promotes collaboration among students, and they are encouraged to engage in social negotiations to deal with the same programming task. With social negotiation and collaboration, the purpose is to communicate different views and ideas and cooperate on problem-solving and knowledge constructing activities (Dabbagh, 2005). Peer support is evident as experienced students actively support younger peers through a collaborative workplace with discussion areas focused on specific topics or programming tasks. Opportunities for collaborative learning provide the ability to learn from others and increase engagement and motivation with the learning process (Looi and Seyal, 2014).

The methodology for identifying threshold concepts is a critical aspect of my research. There are numerous studies on threshold concepts in Computer Science courses. However, most of them cover introductory courses (Khalife, 2006; Eckerdal et al., 2007; Sanders and McCartney, 2016) or teachers' perspectives in secondary education (Kallia and Sentance, 2017). Furthermore, several of the identified threshold concepts mentioned, such as object orientation and pointers (Boustedt et al., 2007), are infrequently used, or their usage is not common in competitive programming competitions. Nevertheless, while the threshold concept framework is highly appealing on a theoretical level, very few programming researchers have attempted to measure their acquisition empirically (Shanahan et al., 2006; Walker, 2013).

The findings of my study contribute to the threshold concept literature in competitive programming. There is minimal data for threshold concepts in my field compared to introductory programming courses. My research provides a methodological approach for identifying threshold concepts based on data from alumni students obtained through questionnaires. I have concluded that dynamic programming, segment trees and recursion are central threshold concepts and can act as theoretical gateways for unlocking students' previously inaccessible ways of thinking about fundamental concepts of the IOI syllabus. The actual effectiveness of these concepts concerning student learning progression was measured by investigating current COI students' actual performance on programming tasks embedded with the identified threshold concepts (Chapter 5).

To measure the level of understanding of threshold concepts, I have established methods of inquiry and evaluation of students' experiences with the threshold concepts. To identify students' strategies and code optimisations, I used the Michanicos code-evaluation platform to collect and analyse rich empirical data. One significant advantage of the Michanicos platform is that it can help teachers identify the liminal variation of a threshold concept based on the students' strategies with the associated tasks. I have measured the variation in students' engagement using qualitative data (strategies, perceptions and source code of students) and quantitative data (accumulated scores from the platform) to assess the level of interaction with the identified threshold concepts.

My research proposes a methodological approach for identifying threshold concepts using data from alumni students and empirically assessing COI students' actual programming performance on these concepts. Only one study (Eckerdal et al., 2007) has investigated programming students in liminal spaces, and they have used semi-structured interviews from a previous study (Boustedt et al., 2007). My research goes further and collects actual performance data from students attempting to solve programming tasks embedded with threshold concepts on a code-evaluation platform. To my knowledge, no other work has explored the liminal spaces of competitive programming students to this extent.

1.2 Competitive Programming

Competitive Programming is about solving complex programming tasks on a computer by writing computer programmes following specific time and memory constraints. These tasks are not research problems, meaning that the tasks' authors have already solved them under the specified time and memory limits. Instead, the competitive component lies in the time frame of the competition, with multiple contestants simultaneously trying to solve the same tasks. To illustrate the general nature of regular programming training material, consider the following task statement of one of the COI practice tasks:

Task: Volleyball teams

Andreas is the administrator of the annual beach volleyball camping event. He is responsible for setting up the teams and assigning each player to a camping tent. Unfortunately, each camping tent can accommodate only one person. Since each beach volleyball team consists of two players who have to practice together, their tents should be as near as possible. Andreas has the coordinates ($-10{,}000 \leq x, y \leq 10{,}000$) of each of the N ($1 < N \leq 16$) tents in a Cartesian coordinate system. He must pair these N tents into K teams of two so that the total distance between all of the teams' tents is minimised.

Sample input	*Sample output*
6 5.0 8.0 4.0 4.0 -6.0 6.0 -10.0 3.0 -2.0 -4.0 4.0 -6.0	15.45

Explanation

Andreas must pair up the players in the following manner creating these three teams. The minimum total distance between the teams' tents is 15.45.

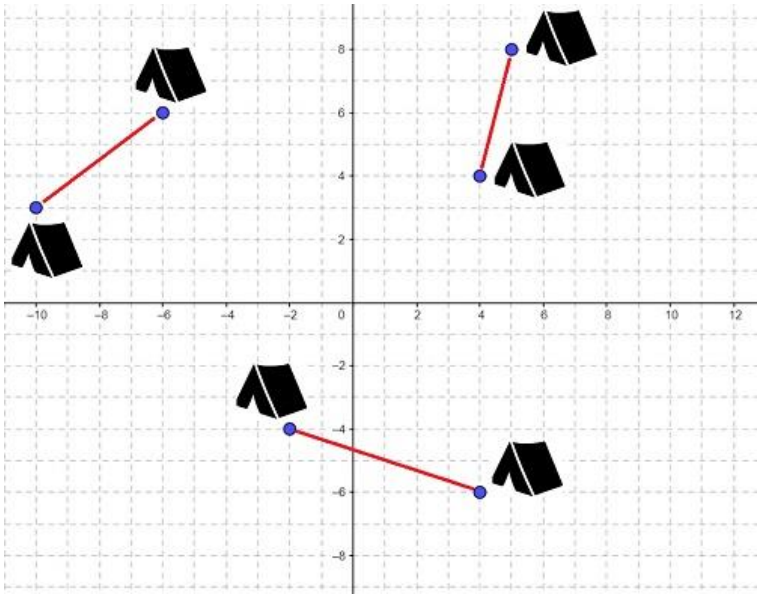


Figure 2: Illustration of the task solution

For most high school or college students, this problem is practically unsolvable by hand due to the large data input set, regardless of their mathematical background, critical thinking capability, or ingenuity. To solve the problem by hand, students must calculate the distances for each possible pair of points (120 total pairs for N=16). Then, they must determine the best arrangement of tents and add their distances to find the minimum calculated total distance. To solve this problem on a computer, high school students must reach a certain required level of programming ability unattainable within current secondary education settings. The students must write code that can produce the same output as the author by using unknown test cases within the allowed time and memory constraints. For a student that has reached the required programming level, this is a trivial minimum-weight perfect matching task, and it is efficiently solvable using dynamic programming[1] (Bellman, 1954). Each state is a bitmask[2] that defines the matching status of each pair, and when unmatched points are matched, it will set the equivalent bits in the bitmask. An experienced student can write the following C++ code that solves the problem in under five minutes.

[1] A technique for solving a problem by breaking it into subproblems and solving each one separately.
[2] A bitmask is a binary representation of the subset of a set.

```
#include <bits/stdc++.h>
using namespace std;
double memo[65536];

struct point {
    double x, y;
} P[20];

double dist(int a, int b) {
    double X = P[a].x-P[b].x;
    double Y = P[a].y-P[b].y;
return sqrt(X*X + Y*Y);
}

int main() {
    int N;
    cin >> N;
    for(int i=0; i<N; i++)
        cin >> P[i].x >> P[i].y;
    for (int p=1; p<(1<<N); p++) {
            memo[p] = INT_MAX;
            int m;
            for (m=0; m<N; m++)
                if (p & (1<<m))
                   break;
            for (int j=m+1; j<N; j++)
                if (p & (1<<j))
                   memo[p] = min(memo[p], dist(m,j) + memo[p^(1<<m)^(1<<j)]);
        }
      cout << fixed << setprecision(2) << memo[(1<<N)-1] << endl;

return 0;
}
```

I have dedicated most of my academic career to using my knowledge to empower others, making high school students capable of solving problems similar and even more complicated than the task 'Volleyball teams' mentioned above. My life's inspiration is to successfully continue challenging arguably some of the brightest minds of future generations. Accordingly, this research project is a personal testament produced within the context of my professional role as a Computer Science (CS) instructor for the Cyprus Olympiad in Informatics and the team leader of Cyprus in international programming competitions.

COI is the learning community that selects and prepares Cypriot delegations that participate every year in the International Olympiad in Informatics (IOI, 2019), the Balkan Olympiad in Informatics (BOI, 2019) and the Junior European/Balkan Olympiad in

Informatics contests (EJOI, 2019). The fundamental mission of COI is to recruit and prepare high school students by increasing their problem-solving skills and performance in competitive programming. My main research objective was to determine students' potential and learning difficulties and create a framework within an evaluative learning environment for improving competitive programming education. I have incorporated multiple theoretical elements into practice to stimulate the desired transformations and encourage new ways of thinking. It was the foundation upon which I formulated the research proposal, the project aims, and the research objectives.

1.3 Purpose and Aim

Programming is the language of Computer Science, and because it is observed universally, it is vital to enhance programming education and effectively promote it to students and society (Verhoeff, 2013). In a world where the essential aptitude employers seek is problem-solving skills (Karzunina et al., 2019), it is critical to support our students to unlock new ways of thinking and understanding. As educators, we must discover, inspire, bring together, challenge and recognise competent students. Accordingly, we ought to improve our teaching practices at all levels of CS education and not just within the IOI community. We need to prepare the new generation of professionals required for the digital economy for the projected 1.5 million new digitised positions worldwide (WEF, 2018). CS education must adapt quickly as the requirements for information and communications technology (ICT) skills in the workforce are increasingly evolving.

The purpose of the study is to empower the next generation of Computer Science experts by helping students comprehend complex programming concepts. The research aims to enhance the pedagogy and the teaching practice of competitive programming education by introducing and assessing a pedagogical framework and utilising a code-evaluation platform. The COI framework intends to redefine the teaching and learning processes within its discipline. To achieve a new way of learning, I must first understand how my current practice is working. Then, I need to improve my knowledge of the teaching practice and pedagogy that I must provide. Finally, aligning with constructivist principles, as discussed further in later chapters, I can understand the role educators play in bringing change to students' lives by learning about them and how they learn.

The most challenging aspect of this project was that I had to teach programming concepts taught in university courses to high school students. These students are not supposed to learn these concepts, no matter how brilliant and competent they are in algorithmic thinking. The learning outcomes model (Hussey and Smith, 2003) states that learning can be achieved with a predefined set of activities to meet specific outcomes phrased as 'by the end of the semester the students must be able to...'. In contradiction, the COI framework challenges students to learn and transform to find their new identity within their learning community. A learning community is a powerful educational practice linked with improved academic performance, enhanced skills, quality knowledge and learner satisfaction with the overall experience (Zhao and Kuh, 2004; West and Williams, 2018). For redefining the teaching/learning processes, I formulated the following research objectives.

1.4 Research objectives

The study has the following research objectives:

RO 1: Investigate and identify threshold concepts in competitive programming and how they relate to the pre-liminal and liminal variations of students' learning.

RO 2: Investigate the process of integrating a framework by the Cyprus Olympiad in Informatics for preparing students for the International Olympiad in Informatics participation and the practical context of this decision.

RO 3: Evaluate the teaching and learning processes of competitive programming using a code-evaluation platform with competition-type programming tasks embedded with identified threshold concepts and measure the effect on students' strategies.

RO 4: Determine if the degree of related student engagement and motivation with the learning process can improve the learning outcomes for the Cyprus Olympiad in Informatics.

To meet the research objectives, I chose to combine action research with mixed methods (Maxcy, 2003; Johnson and Onwuegbuzie, 2004; Thota et al., 2012). Mixed methods require a hands-on, performance-oriented focus for incorporating quantitative and qualitative practices with action research methodology. Combining approaches and methods can be accomplished by sharing the findings obtained from quantitative data

using action learning sets and focus groups. The data will be applied to support reflection, decision-making and further actions (Parker et al., 2017). It has been reported that the combination of the two can produce scientifically robust and transferable results to instruct planning, implementation, evaluation and adjustment (Ivankova and Wingo, 2018). The following is an overview of the empirical study and the methods used with each of the research objectives:

Research Objective	Research Purpose	Data Source	Form of Analysis
RO1	Qualitative evaluation, identifying threshold concepts	Questionnaires Interviews	Qualitative analysis
RO2	Qualitative, quantitative evaluation	Action research	Qualitative, quantitative analysis
RO3	Qualitative, quantitative evaluation	Platform data	Qualitative, quantitative, statistical analysis
RO4	Quantitative evaluation	Questionnaires Action research	Quantitative analysis

Table 1: Overview of methods used in the empirical study

Cyprus Olympiad in Informatics supports future computer scientists for a world with a growing dependence on technology and its applications in our lives. Confidently, my research will improve the quality of competitive programming education, deliver steadily increased performance for Cypriot delegations in international contests and establish the COI framework as a distinctive contribution to the IOI community.

1.5 Chapter summary

This chapter presents the motivational context in which I have considered my research proposal, aim and objectives. The research aim is to enhance the teaching practice of competitive programming education. The research objectives are the foundation for evaluating the proposed framework's impact and bringing forth this research's distinctive contributions.

CHAPTER 2: Review of Knowledge and Information

2.1 Introduction

This chapter presents a review of knowledge and information on investigating the theoretical and practical features of competitive programming education. The literature review covers pedagogies and methodologies used for the instruction of students in programming courses and ways to increase students' problem-solving skills and performance. Based on the literature, I focused on particular features that have been proven to be effective in programming practice. I reviewed pedagogical practices to enhance students' programming strategies and deal with the conceptual difficulties. To put everything into context, I also present an introduction to the educational system of Cyprus, the Cyprus Olympiad in Informatics (COI) and the associations with the stakeholders and policymakers of the COI.

For this research project, I have investigated the theory of threshold concepts, how it can be applied in competitive programming and the ways that threshold concepts are affecting student learning. I was very interested in exploring liminality and the distinct states a student is navigating through the learning process. Additionally, in this chapter, I have studied the principles of framework design, the usefulness of mental models, the significance of student strategies, the effects of competitions on engagement and motivation, and the importance of online judges in programming education and their effects on students' performance.

My goal is to make competitive programming meaningful and accessible for students and improve programming education within my discipline. Therefore, a framework that encompasses all of the above features is in great need. Such a framework should be meticulously designed, and with further research, it can become readily accessible in other levels of programming education. As students' abilities increase, the need to extend the possibilities of student instruction outside school settings is much more prominent (Morelock and Feldman, 2003). An introduction to the educational system of Cyprus follows next.

2.2 Computer Science in the Educational System of Cyprus

Computers were introduced to secondary education in Cyprus in the early 1990s as part of technological advancement (MOEC, 2013). In 2001, with the establishment of the unified lyceum, this initiative received more support. Consequently, with the induction of Cyprus to the European Union (EU) in 2004, numerous EU funding and projects were used for modernising the school equipment and for teacher preparation. As a result, in the European Commission survey (EC, 2013), the schools of Cyprus were rated higher than the EU average, with exceptional internet accessibility and connection speeds. Secondary education is divided into two parts: The first part is the gymnasium, a compulsory three-year period, and the second is the lyceum. This voluntary three-year period leads to entry examinations for tertiary education (Figure 3).

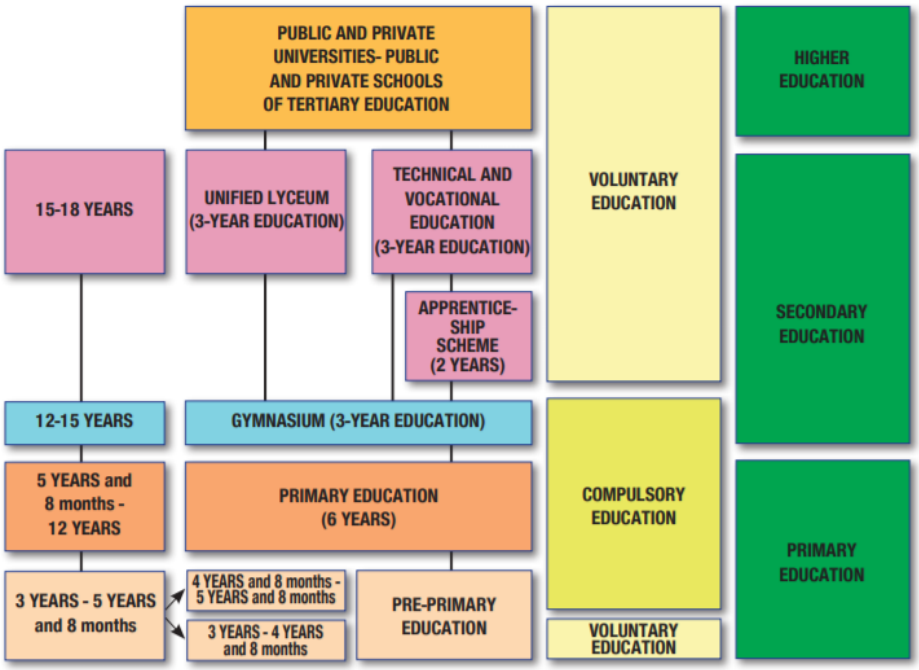


Figure 3: The Educational System of Cyprus in 2019

Today, CS courses are taught as follows: First, the compulsory course 'Computers' in the gymnasium for ages 12 to 15 for two hours per week (Figure 4). Then, the mandatory successor course in the lyceum for age 16 for two hours per week. Last, the four elective courses for the second and third grades for ages 17 and 18, respectively. Currently, these electives include two distinct directions for each class: The enrichment form (algorithmic-

oriented) and the applications form (application-oriented), each of them taught for four hours per week.

GYMNASIUM									
		GRADE 1		GRADE 2		GRADE 3		TOTAL FOR 3 YEARS	
FIELD	SUBJECT	PERIODS	%	PERIODS	%	PERIODS	%	PERIODS	%
COMPUTER SCIENCE		5	13.5	4	10.8	4	10.8	13	11.7
	Computers	2		2		2		6	5.4

Figure 4: Computers course periods per week in the gymnasium

CS is widely considered a necessary skill, and many global initiatives such as 'Computer Science for All' (CSforALL, 2019) aim to make the discipline an integral part of high school education. However, no equivalent course is currently in the common core subjects for lyceums in Cyprus (Figure 5).

Common Core Subjects at Unified Lyceum		
SUBJECT	FORM B Periods	FORM C Periods
MODERN GREEK	4	5
ANCIENT GREEK	1	0
MATHEMATICS	3	2
NATURAL STUDIES	3	0
Physics	2	0
Chemistry	1	0
Biology	0	0
Natural science	0	0
HISTORY	1.5	2
RELIGIOUS STUDIES	1.5	2
CIVICS	0	1
ECONOMICS	0	0
COMPULSORY FOREIGN LANGUAGES	4	4
English	*	*
French	*	*
FOREIGN LANGUAGE 1	2	2
FOREIGN LANGUAGE 2	2	2
COMPUTER SCIENCE	0	0
TECHNOLOGY	0	0
PHYSICAL EDUCATION	1	1
MUSIC	0	0
ART	0	0
Total number of periods	**19**	**17**

Figure 5: Lyceum Computer Science core subjects

From my experience, the courses above, even though revised, are inadequate to produce better programmers with the current teaching practices. Also, the programming curriculum is very restricted and unfitting to prepare our students for programming competitions. Regrettably, the lyceum practices employed for programming education currently include writing source code on paper.

To promote programming education outside of the school, CS educators have organised programming contests since the initial introduction to students. Undoubtedly, education and competitions are firmly connected, especially when contemplating contests as a means to endorse and strengthen programming education (Dagiene and Skupiene, 2004). Dagiene also reported on competitions for learning competitive programming in secondary schools (Dagiene and Skupiene, 2004) and for general education (Dagiene, 2005). Audrito (2012) reports that competitions have a considerable impact on programming education as they create an inclination to begin programming education as early as possible. Therefore, competitions must be an indispensable aspect of education (Verhoeff, 1997).

Additionally, Pohl (2006) reports that when their involvement and achievements in competitions are distributed, it boosts students' enthusiasm and motivation. To send a team to international contests, Cyprus has to organise a local competition for selecting its best students initially. The formula for administering the local competitions is the responsibility of the participating countries, given that objectivity is secured between the participants. The Cyprus Olympiad in Informatics supported this procedure (COI, 2019).

2.3 Cyprus Olympiad in Informatics (COI)

In 1990, high school competitions in maths and physics with unusual challenges marginally above the regular school syllabus were labelled 'Olympiads'. The Ministry of Education of Cyprus (MOEC), in cooperation with the Cyprus Computer Society (CCS), established the Cyprus Olympiad in Informatics (COI) in 2004. Since then, the COI has offered weekly lectures and annual camps for high school and elementary students. In partnership with the MOEC and the CCS, the COI organises an annual competition resulting in forming the Cypriot delegations that participate in the IOI, the BOI and the EJOI/JBOI competitions.

COI has substantially grown since it was founded. At first, there was an initial theoretic round involving writing algorithms on paper and then a coding round on a computer. Next, the setup of two coding rounds with separate time frames was applied. Significant variations were established, such as the number of points awarded for solved tasks or introducing numerous subtasks for some tasks. Presently, the COI is associated with international competitions and follows specific guidelines and curricula.

Currently, the COI is organising four programming competition rounds. The competitions' scores are evaluated by the CMS (CMS, 2019), the identical Contest Management System used in IOI and BOI. The Bebras competition is organised annually to attract COI newcomers and is the first level for COI participation (Bebras, 2019). The competition requires problem-solving skills rather than programming skills. Participating in the Bebras contest is not mandatory, and it does not have a qualifying objective. It merely measures student readiness to compete in the COI competition rounds. Junior and senior levels are the next two levels of participation for COI. Despite the formal age limit of fifteen for the advanced level, younger students can participate if they display extraordinary programming abilities. For example, the youngest contestant in 2019 was eleven years old. Additionally, several colleagues participate in the second-round contests, confirming that COI is a learning community for all individuals.

2.4 The COI Community

The International Olympiad in Informatics (IOI) is the most prestigious programming contest globally. The United Nations Educational, Scientific and Cultural Organisation (UNESCO) and the International Federation for Information Processing are patrons (IOI, 2019). The IOI was organised for the first time in Bulgaria in 1989 and by other countries annually ever since. The competition symbolises the highlight of programming for secondary education.

Approximately 900 participants, including 350 contestants from 85 countries, participated in the IOI 2018 in Japan to solve six programming tasks in two competition days (IOI Statistics, 2019). Most of the participating students were familiar with programming concepts taught in university CS curricula. Cyprus initially took part in 1993 and became a consistent participant. With an expanding support base from scholars, educators and alumni, an international movement was created to support the IOI. I had the opportunity to participate in the former five IOIs (2015-2019) as the team leader of Cyprus (IOI Statistics, 2019).

The Balkan Olympiad in Informatics (BOI) is the equivalent of IOI for students of the Balkan region and adjoining nations: Cyprus, Greece, Bulgaria, Bosnia and Herzegovina,

Croatia, North Macedonia, Albania, Italy, Montenegro, Romania, Moldova, Slovenia, Serbia and Switzerland. Like the IOI, the BOI has two competition days and a practice day. On each competition day, the contestants are presented with three programming tasks to solve within a time frame of five hours. The students programme on a computer without any help, particularly without interaction with team members, contestants, or reviewing notes and books.

To solve a problem, the contestants have to create a programme using any permissible programming languages and submit it through the system within the time and memory constraints. The source code is evaluated with unknown test cases and graded accordingly. Several tasks are separated into subtasks with increasing complexity, and points are won when all of the subtask's test cases produce accurate results. The scores from both days are accumulated for every participant. Depending on their total score, contestants are awarded medals. Approximately the top half of students get medals, so the gold: silver: bronze: no-medal ratio is about 1:2:3:6 (IOI, 2019).

Cyprus organised the BOI of 2016. It has been an extremely challenging assignment that verified our capabilities to coordinate and manage such a significant event. By being in the scientific committee of the competition, my duties included: task setting and assessment, delegation training, hosting coding camps, CMS server administration, all similar to my job description. What we at COI have achieved with hosting BOI 2016 was that we received more interest from our stakeholders: the MOEC and the CCS.

The Ministry of Education and Culture (MOEC, 2018) is the supervisor of COI. The Ministry is responsible for teacher selection and assistance and sets the directives for all the participants. Additionally, it establishes the specifications for the CS syllabus for secondary education. The CCS is a non-profit organisation that recognises the effect of CS on academics, employment and society, and on the quality of life of individuals. The yearly COI budget is minimal, and the CCS covers up to 70% of our annual expenses required for travelling. Government resources cover the remaining costs through the MOEC.

The theoretical background of the study follows next. My motivational context for investigating the teaching and learning processes of competitive programming involves pedagogical, academic, and practical factors.

2.5 Teaching and learning programming

Competitive programming is much more complicated than introductory programming because it is more of an art than a body of knowledge. Most of the concepts that high-school students must learn can be found in university curricula. Writing a computer programme for solving a problem is equivalent to writing poetry or composing music (Knuth, 1974). Knuth suggested that there is no appropriate style for writing a programme as students have unique styles and should not be forced into an unnatural mould. From my experience teaching competitive programming for the past decade, programming is best learned with practical engagement with complex programming tasks. Each programming task can be a special case of a problem-solving session where students can solve at their own pace, with regular feedback from teachers and peer support within a learning community (Wilson et al., 1996).

The implementation and teaching of a programming course should be realistic in its expectations and systematic in its development. Multiple pedagogical approaches that consider learning theories and information technology have been introduced for introductory programming. However, there seems to be a lack of agreement on the best method to teach programming (Robins, 2019). Furthermore, advances in Computer Science have led to more different methods in introductory programming courses (ACM, 2013). Moreover, the teaching approaches used in these courses are in a greater state of variation (Robins, 2019).

Despite the disagreement on applicable specifications, there is agreement on guidelines and theoretical issues arising from the experiences of instructors and Computing Education Research (CER). In a literature review on teaching programming, Pears et al. (2007) propose three approaches based on the central focus of the instructional design: problem-solving, discovering a specific programming language, and code development. Linn and Dalbey (1989) suggest an ideal sequence of cognitive achievements for teaching

and learning programming. The sequence includes programming language features, design skills that include knowledge schemata, planning, testing, and optimising code, and problem-solving skills that include knowledge, strategies and aptitudes of the specific language that can be used for unknown problems and circumstances.

Acknowledging the significance of problem-solving, numerous ACM course standards address it along with the programming language aspects (ACM, 2013). Multiple studies report improved results with this method (Davies, 2008; Koulouri et al., 2014; Hill, 2016). Kay et al. (2000) report considerable progress in programming competency in reviewing the issues involved in problem-based learning. The connection between problem-solving and programming skills is broadly reviewed by Palumbo (1990), and a report of numerous instances is explained.

Robins et al. (2003), in a broad review of the literature, used three dimensions to propose an outline for course design and implementation: knowledge, strategies and mental models. Learning to programme involves acquiring the necessary declarative knowledge (understanding how recursion works) and using practical strategies for its application (using recursion in a programme) (Davies, 1993). Making the dimensions explicit to students may engage them in the learning process and support their understanding. Successful learning of the three dimensions depends on engaging with programming tasks because mental models, programming knowledge and skills cannot be efficiently developed in theory; they have to be rooted in practical experience (Robins, 2019).

Research on novice programmers has indicated that the core challenges are not related to the programming language concepts but to general programme structure and design (Lahtinen et al., 2005; Garner et al., 2005; Robins et al., 2006). This issue is consistent with numerous suggestions in the literature that teaching should focus on combining language elements with the core issue of programme design. Spohrer and Soloway (1989) recommend concentrating specifically on strategies to integrate and coordinate the concepts beneath the source code. Students should acknowledge these concepts and be equipped with new ways of writing programmes. Specifying and presenting basic

schemata or coding patterns should be introduced in computer programming curricula (Mayer, 1989).

A review of programming education research involves an approach to teaching which utilises a shift from studying the syntax of programming languages to the progression of overall problem-solving and code-writing abilities (Caspersen and Bennedsen, 2007). Moreover, the review underlines the value of worked examples, along with scaffolding, controlled feedback, and emphasis on patterns to support schema creation and improved learning. Learning schemata involves mindful abstraction, assumes the confrontation with a carefully selected set of tasks and their worked examples, and offers comparisons that could guide successive performance in solving unknown programming tasks (Van Merrienboer and Paas, 1990). Worked examples or completed programmes are supportive and available sources of information, but the strategies that produced them are more challenging to define.

Robins et al. (2003) suggest that strategies are the most critical aspect for defining failure or success of learning to programme. The authors propose that differences in strategy distinguish effective and ineffective learners. Regardless of their significance, strategies are not considered as much as knowledge in programming courses. Brooks (1990) suggests that programmers' strategies cannot be determined from the final form of a programme, even if they might affect the coding process and thus the final form. A critical point that arises here is the possibility to assess the coding process of students from their initial submission to the final one. Continuous assessment of students' code throughout the coding process can support code optimisations and identify optimal strategies for solving complex tasks. The latter highlights the significance of actively creating programmes and explicitly focusing on the strategies used in course design and instruction.

Like schemata, mental models are adopted from cognitive science and have been variously defined and widely utilised (Johnson-Laird, 1983; Gentner, 2002). The mental models that students must create, according to a fundamental notional machine (Du Boulay, 1986) and the critical features of planning, understanding, and optimising code, are vital for

improving learning. Mental models of specific programmes or algorithms are internal and unique to students and can be linked to particular examples. Therefore, pedagogical and technical features of course design should implicitly support students in the acquisition process of mental models by providing worked examples and encouraging the process of optimising code.

Multiple implications for practice can emerge from the literature for designing a programming course and developing a pedagogical framework to foster it. I have chosen to familiarise myself with most of the research, adopt the specific methods that have been proven to be effective in programming practice and be as reflective as possible during the study. I have considered these particular methods in the context of pedagogical models, conceptual difficulties, problem-solving, motivation, scaffolding, collaboration, assessment, and feedback. Nevertheless, the most effective teaching method related to these features, and the best practices to support programming students, are still elusive.

Most of the proven methods in the literature have been found to fit with the situated cognition view of knowledge. In situated cognition, learning is defined as a process of constructing meaning from activity and experiences (Jonassen, 1991) and is consistent with the epistemological orientation of constructivism. Knowledge is acquired through facilitated forms of interaction through a community of learning and practice where the students actively negotiate their understanding with the external world (Dabbagh, 2005). The students take ownership of their learning and are the primary meaning-makers, while the teacher is the facilitator of learning who provides scaffolding and creates an innovative learning environment. Lave and Wenger's work on situated learning (1991) has been decoded by Meyer and Land (2006) into a space that lies between basic concepts with marginal involvement and threshold concepts, which allow full participation in the community of practice (Walker, 2013).

Based on situated cognition and constructivist views of learning, the emerging pedagogical models and instructional strategies provide the instructional sequence control to students (Coleman et al., 1997) and allow them to construct depictions of particular meaning (Hannafin, 1992). Pedagogical models that embody these attributes include communities

of practice, knowledge-building communities and learning communities. Instructional strategies that exemplify these characteristics include facilitating problem-based learning, promoting authentic learning activities, promoting social negotiations and collaboration, and providing scaffolding.

2.6 Pedagogical framework

Pedagogical frameworks define the general rules through which theory is employed in the teaching and learning processes. Bednar et al. (1991) indicated the significance of connecting theory to practice for developing any instructional design and emphasised that efficient design is achievable when the creator has an intuitive understanding of the design's theoretical foundation. Hadjerrouit (2008) describes a pedagogical framework as an awareness of the connections between learning theories, educational practices, information technology, and an academic discipline. Ivala et al. (2013) claim that frameworks present a method for integrating a discipline's pedagogical model into the learning environment.

Modern advances in web-based technologies have redefined the limitations and extended the scope of such frameworks by deepening their interconnectedness (Dabbagh and Bannan-Ritland, 2005). Moreover, all educational learning environments must be embedded in epistemological frameworks to be efficient for teaching and learning. Therefore, describing learning and creating guidelines for programming education must include suitable instructional structures and appropriate online learning technologies and tools to promote student learning.

Pedagogical approaches, which exploit learning theories and technologies, have been introduced in the literature to deal with the learning obstacles associated with programming education (McGowan, 2016). Dabbagh's framework (2005) is based on two other studies. Coleman's research (1997) on requirements of pedagogical models emerging from constructivist views of learning. Hannafin's research (1992) for assigning students with creating depictions of personal meaning.

Dabbagh (2005) suggests that three key components are working together to promote meaningful learning and communication. These are pedagogical models, teaching and learning strategies and online learning technologies or pedagogical tools. The three elements form an iterative relationship. The models, grounded in constructivist theory, inform the framework design by specifying the instructional strategies facilitated by learning technologies (Figure 7). I have used these three components for the development of the COI framework.

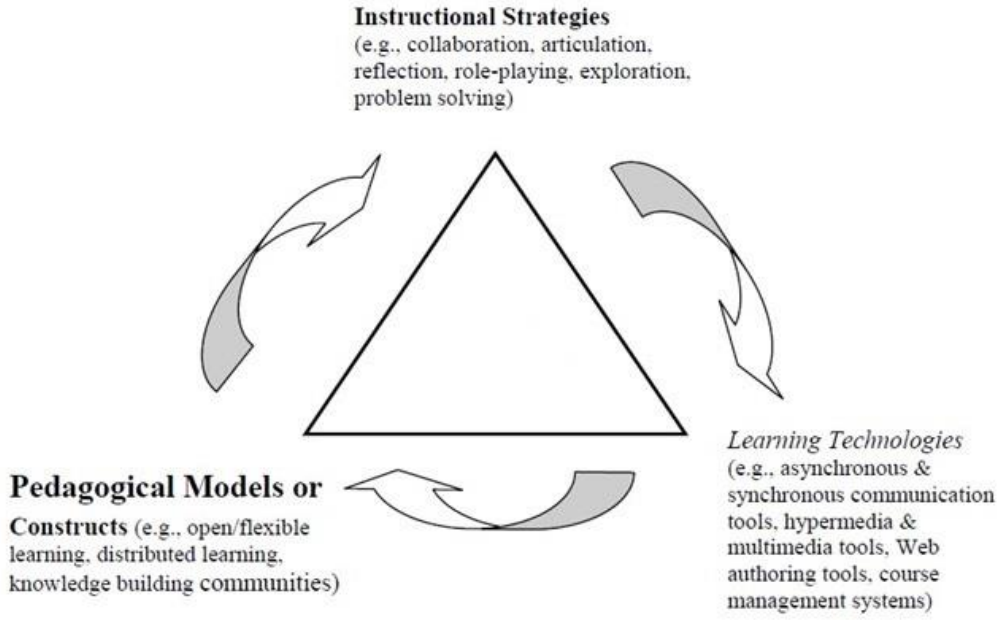


Figure 6: Theory-Based Pedagogical Framework design (Dabbagh, 2005)

The first component of the COI framework design is the pedagogical models. The pedagogical models are theoretical constructs that originate from knowledge acquisition views about cognition and are the basis of the learning theory. These models are the methods to associate theory with practice. Conole (2010) presents a review of pedagogical models that have been used in education based on the following three learning theories as they are grouped by Mayes et al. (2004):

- Associative learning as an activity through structured tasks
- Cognitive learning through understanding
- Situative learning as social practice

Mayes et al. (2004) emphasise that learning theories are empirically based accounts of the parameters which impact the learning process and consistently support the instructional design planning. The theoretical underpinnings use one or a combination of learning theories to describe views about cognition and the construction of knowledge (McGowan, 2016). Consequently, these viewpoints shape the beginning of the COI framework designed to communicate threshold concepts of competitive programming.

To link the COI framework with existing education theory and establish its theoretical underpinnings, I reviewed Conole's report (2010). Conole presents twenty pedagogical frameworks and models, all of which have online components. Thirteen of these frameworks are classified according to whether they adopt associative, cognitive or situative learning perspectives. Five are categorised as generic, and two are mainly about assessment practice. Conole also considers several advantages of adopting pedagogical models and mentions several constraints.

Pedagogical models can be used as guidelines or schema to align a specific pedagogical approach. Moreover, they can steer design decisions about the learning activities that would encourage the pedagogical method used in the framework. Similarly, they can be applied to support the design of a learning environment. There has been some criticism of pedagogical models because they are theoretical constructs. Instructors may misinterpret how to employ the framework successfully by implementing a superficial application of the model to their practice (Lisewski et al., 2003). The elements involved in learning and teaching and their interdependencies show that pedagogical frameworks are not a panacea or a shortcut to a coherent structure (Conole, 2008).

The associative perspective concentrates on performance adjustment via stimulator-reaction pairs, trial and error learning, learning through association/reinforcement, and measurable results. The most influential theoretical approach supporting this theory is an instructional design based on the deconstruction of learning (Gagne, 1973) into elements intended to develop knowledge and skills throughout a sequence of processes. Merrill examined instructional design models and theories and described a set of interrelated prescriptive instructional design principles (Merrill, 2002). Merrill's model implies that

problem-based settings are the most efficient learning environments in which students are engaged in four different phases: activation of previous knowledge, presentation of skills, application of skills and incorporation into real-world activities. Collis and Margaryan (2005) have added six related standards concerning the successful application in specific learning environments: technology support, supervisor support, reuse, collaboration, differentiation and learning from others.

The cognitive perspective identifies learning as a transformation in core cognitive structures. Educationally, it is illustrated by administering and communicating information through problem-solving, explanation, communication, recombination, contrast and inference (Driscoll, 1994). It generates constructivist and reflective views. An example of a framework that supports constructivism was created by Jonassen (1999; 2003). It can be used as a guideline to develop Constructivist Learning Environments (CLEs). A CLE is an environment where students may assist others using various tools and information resources in their quest for learning objectives and problem-solving assignments (Savery et al., 1995; Wilson, 1996; Gance, 2002). The crucial claim is that learning happens when students actively engage in making meaning.

The situative perspective defines learning as social participation and highlights interpersonal relationships involving modelling, imitation, and the collaborative construction of knowledge. Knowledge in this perspective is viewed as belonging and allocated in communities of learning. The learners practice the patterns of learning and inquiry to become members of the community (Firdyiwek, 1999). In educational settings, such communities can be groups of students that share resources and information, solve tasks and accomplish common goals, and by doing so, collaboratively acquire new knowledge and advance the methods of the community (Dabbagh, 2005). Pedagogical models include Wenger's theory of communities of practice (1998), firmly embedded in the situative perspective. However, the Activity theory also encompasses some aspects of the cognitive perspective (Conole, 2010). Activity theory initiates on the assumption that activities occur within a context that needs to be considered to make the situation meaningful and appropriately interpret the results (Mwanza, 2002).

Even though it was not explicitly developed as an educational environment, Wenger's theory of communities of practice is invaluable as it examines how communities of practice are established. Wenger identifies four main attributes of learning: a community, an identity, a meaning, and a practice (1998). Therefore, each one is important because it emphasises particular attributes of learning, which can then be used to offer support. Communities of practice and knowledge building communities (KBC) are synonymous concepts. However, learning communities might be viewed as a broader term that involves any social group or structure that brings people together to pursue and share knowledge (Dabbagh, 2005).

Learning communities, such as the COI community, are groups of individuals who support each other in their learning journeys, work collectively for solving problems and learn from others and their settings. Students and teachers engage in a collaborative practice where participation transforms into new experiences and new learning (Rogoff, 1994; Wilson et al., 1996). Learning communities exemplify a deliberate reform of students' learning experiences across an interdisciplinary premise to encourage specific emotional and intellectual relationships among students, teachers, and disciplines (MacGregor et al., 1999). Learning communities perform as social and academic support constructs that stimulate students to engage in more challenging and authentic learning methods. They are identified as informal learning environments, shifting the focus from teaching to learning. Preece (2001) proposed a framework for forming and supporting online learning communities based on two crucial components: usability and sociability (Preece, 2001). The components can determine the design criteria and associate success factors.

Teaching strategies is the second component of the COI framework design. Strategies are what teachers or education systems do to facilitate student learning. Jonassen et al. (1991) describe instructional strategies as teaching methods to engage students and facilitate learning. When discussing the implications of the learning theories for education, teaching strategies are the essentials of how these inferences will be transformed into instructional processes (Shuell, 1980), resulting in a method, or sequence of activities, intended for achieving a specific objective (Jonassen et al., 1991). Therefore, instructional

strategies originated from pedagogical models, which, in turn, originated from learning theory.

Instances of instructional strategies that exemplify the attributes of pedagogical models grounded in constructivist views involve: supporting authentic learning activities, enabling problem-solving, promoting collaboration and providing scaffolding. Generally, the purpose of instructional strategies is to deliver a learning environment where learning with self-awareness, collaboration, and self-management are encouraged. The teacher's role is supportive, reciprocal, communicative and responsive to the students' requirements (McLoughlin et al., 1999).

Problem-Based Learning is an instructional approach that is learner-centred, and students learn by solving problems (Barrows, 2000; Hmelo-Silver, 2004). Within the COI framework, students must identify what they have to understand to solve a problem. They take part in self-directed learning and then employ their acquired knowledge to solve the task, reflect on what they learned and the efficacy of the strategies they used. The teacher acts as a facilitator to the learning process rather than the knowledge provider.

PBL supports knowledge construction as students activate their previous knowledge in their initial negotiations (Schmidt et al., 1989). It also supports the social construction of knowledge as students work in small teams using analytical skills to solve real-world problems (Greeno et al., 2006). From a cognitive perspective, organised learning experiences support students' understanding of concepts through problem-solving activities. However, from a situative perspective, social interactions and negotiations are the sources of knowledge construction (Hmelo-Silver, 2012). This viewpoint recognises that social practices assist the growth of students as capable learners and proficient both as problem solvers and in their disciplinary knowledge (Lampert, 2001).

One approach for promoting a constructive environment that has been broadly employed in developing learning environments is cognitive scaffolding. The learning activities are reinforced by a series of guidelines (offering hints, reminders, and feedback). Scaffolding supports students to reflect on their actions and helps them develop metacognitive skills.

As these skills develop, the scaffolding is gradually removed. There is a vast amount of research on scaffolding and learning in problem-based environments (Collins et al., 1989; Davis et al., 2000; Golan et al., 2002) and programming courses (Vihavainen et al., 2013). The third component of COI framework design is the learning technologies. Research on the instructional usage of technology in pedagogical frameworks has revealed that instructors in higher education do not have the necessary knowledge to integrate technology into their teaching practices effectively, and their efforts are likely to be limited in scope and depth (Kochler, 2013). The COI framework introduces two such technologies: a contest management system (CMS, 2019) to organise and administer programming competitions and an online code-evaluation platform (Michanicos, 2018) used in the COI course. The contest management system and the automated code-evaluation platform offer multiple advantages to a programming learning community. These will be presented more extensively in the following chapters.

Implementation of the COI framework requires an authentic learning community to incorporate the teaching practice and the learning technologies. A learning community where the knowledge constructed is meaningful for the student, relates to the world outside the classroom, offers an opportunity to reflect in the modes of the discipline, and where the means of assessment reflect the learning process (Shaffer and Resnick, 1999). Connolly and Berg (2006) have recommended that the term authentic should be expanded to include learning using the tools and methods of contemporary professionals.

However, learning to programme is generally acknowledged to be very challenging. Multiple reports indicated that many students completed CS courses having fragile learning, without a firm grasp of fundamental concepts in the past four decades (Soloway et al., 1983; Kurland and Pea, 1989; McCracken et al., 2001; Lister et al., 2004; Utting et al., 2013; Robins, 2019). Students must have the proper conceptual understanding of a notion and employ it in a concrete approach using appropriate strategies (Robins et al., 2003; Lahtinen et al., 2005). The latter also supports the assumption that the major challenge is not the programming languages used but rather the structures and concepts and the required new way of thinking. When students go through a transitional period to understand this new way of thinking, it will enable them to implement optimal

programming strategies designed using this new way of thinking and acquire a concrete understanding of concepts.

According to the Threshold Concept theory, students get stuck in these transitional periods or liminal spaces. Meyer and Land (2005) have proposed using threshold concepts to differentiate particular topics that might be used to organise the learning process. They further developed a theoretical framework, the liminal space, specifically focusing on learning these concepts. Eckerdal et al. (2007) identified a framework for determining when and where a student is in a liminal space while learning to programme. Eckerdal's framework was based on dissimilar types of understanding: abstract understanding, concrete understanding demonstrated through practical programming, the aptitude to progress from abstract to concrete understanding, recognising why the concept is used and acknowledging the application of the concept in new conditions (Eckerdal et al., 2007).

The COI framework builds upon all of these theoretical underpinnings. It recognises the conceptual difficulties of programming students, identifies threshold concepts in competitive programming, and proposes learning technologies to foster the pedagogical model of a learning community. Moreover, it uses Problem-Based Learning as an instructional method, provides scaffolding with three phases of a redesigned curriculum, promotes motivation with competition participation and encourages peer support and collaboration. All of these features are discussed more extensively in the following sections.

2.7 Conceptual challenges of programming

Very few programming educators would argue that students find writing programming code easy. Many of us are familiar with the struggles and disappointments of novice programmers as they struggle to comprehend even the most basic coding patterns. Decades of research have informed that learning to programme is a complicated process for many students. Hagan et al. (1997) noted this concern, reporting that programming was considered the most challenging and least exciting subject in all computer science courses. Many authors have supported the claim that programming is a complicated task

throughout the years (Baldwin and Kuljis, 2001; Jenkins, 2002; Robins et al., 2003; Hanks et al., 2004; Bergin and Reilly, 2005; Lahtinen et al., 2005; Gomes and Mendes, 2007; Butler and Morgan, 2007; Konecki, 2014).

Consistent with the learning difficulties surrounding programming, studies have shown that failure and dropout rates are high in introductory programming courses (Nikula et al., 2011; Teague, 2011; Mendes et al., 2012; Watson and Li, 2014). In a survey of 63 institutions internationally, the pass rates of programming courses were estimated to be approximately 67% based on group size and other factors (Bennedsen and Caspersen, 2007). Learning programming requires problem-solving skills that are complicated and multi-dimensional. Programming is complex because the skill-set includes particular cognitive skills that are not used in everyday life or work activities. It is multi-dimensional because it also requires an understanding of the syntax and semantics of a specific programming language. It also requires an understanding of fundamental programming concepts such as conditions, loops and functions. In the end, the learner has to apply all of these aptitudes to solve a complex problem.

Du Boulay et al. (1981) describe five overlapping domains that must be learned by a novice programming student: what programmes are for and what can be done with them; a general model of the computer as it relates to the execution of programmes; the syntax and semantics of a specific programming language; the use of schemata for organising knowledge; the skills of planning, developing, testing and debugging programming code. Although most programming instruction focuses on the syntax and semantics of programming languages, novice programmers will generally deal with many of these domains at once, increasing the difficulties.

In their literature review on novice programmers, Robins et al. (2003) synopsised the investigated issues using the domains of knowledge, strategies and models (Figure 7). The columns describe the qualities required for writing code: knowledge of a programming language; strategies for using this knowledge correctly; ability to create and evaluate mental models of programme states. The rows define the stages of developing a programme: the processes of design, generation (code-writing), and evaluation. Similar

to Du Boulay's domains, a novice programmer will generally deal with several of these requirements simultaneously.

	Knowledge	***Strategies***	***Models***
Design	of planning methods, algorithm design, formal methods	for planning, problem-solving, designing algorithms	of problem domain, notional machine
Generation	of language, libraries, environment/tools	for implementing algorithms, coding, accessing knowledge	of desired programs
Evaluation	of debugging tools and methods	for testing, debugging, tracking / tracing, repair	of actual program

Figure 7: A programming framework by Robins et al., 2003

When considering the requirements, the essential capabilities and the interactive nature of programming, I must also investigate the constraints that apply to the teaching process. It is, therefore, necessary to acknowledge the difficulties that programming students encounter. Lahtinen et al. (2005) surveyed 559 programming students and 34 teachers. Students identified the most challenging aspects of programming to be understanding how to design a programme to solve a particular task and discovering bugs (errors) in their programmes. These issues do not relate to the syntax and semantics of the programming language but instead create correct mental models and strategies (Lahtinen et al., 2005). The authors suggest that the biggest problem of novice programmers is not the understanding of basic concepts but mastering how to implement them correctly. In the same study, students found example programmes to be the most useful material for learning to programme.

Similar studies explored the problems encountered by novice students attempting programming tasks over two successive years (Garner et al., 2005; Robins et al., 2006). The most repeatedly recorded issues were: understanding the task, issues relating to overall programme design and structure, and basic procedure. Similar to the Lahtinen study, these studies suggest that developing the algorithm and the overall programme design is more complex than implementing any particular programming language construct.

Butler and Morgan (2007) reported that programme design elements were among the most complex introductory programming curriculum aspects for approximately 150 students. Curriculum concepts such as object orientation and algorithms with a relatively high conceptual nature proved challenging both from an understanding and an implementation perspective. The same study also discovered a shift in perceived difficulty from understanding to implementing most curriculum concepts. The only concept for which the shift was not experienced was the syntax and semantics of the programming language (Butler and Morgan, 2007).

Butler and Morgan (2007) also reported that students experienced conceptual difficulties with curriculum concepts that required abstract and logical thinking. Very little feedback was available, and student performance on these concepts was poor. The authors further suggest that feedback is inherently limited in programming environments. Therefore, consideration should be given to teaching methods that provide quality feedback both inside and outside the classroom. Moreover, the teaching method should scaffold the learning and support the student learning through the programme design to reduce the perceived complexity of the troublesome concepts (Butler and Morgan, 2007).

Jenkins (2002) argues that two cognitive factors might complicate learning programming: learning style and motivation. As students tend to learn in different ways, some may prefer a more solitary approach, and others may prefer to learn by discussing with their peers. Students tend to have a variety of motivations regarding programming (intrinsic, social and extrinsic) and the students who struggle most tend to have primarily extrinsic motivation (Jenkins, 2002). I will look into motivational issues in the following sections.

Konecki (2014) surveyed 190 students after their programming courses. He also drew attention to motivation and appropriate learning styles and suggested that programming courses integrate different aspects of various learning styles to be suitable for all programming students. Programming students have a specific intuitive understanding of programming concepts based on their previous knowledge, age and experience (Pea et al., 1983). However, this intuitive way of thinking might be the main reason for most of their struggles (Konecki, 2014). Konecki concludes that some form of constructivism

should be applied when designing programming courses as new methods that promote this way of thinking are required. The teacher should act as a facilitator and integrate elements of constructivism into the teaching process that will support students to understand the concepts by themselves. In this way, the students will acquire a concrete and durable understanding rather than a superficial one.

One way of investigating the challenges faced by novice programmers is to compare them to experts. The widely agreed belief is that the most crucial distinction between experts and novices is the richness of their individual experiences or collections of learned schemata (Robins, 2019). Additionally, a key factor separating novices and experts is the effectiveness of the strategies they are using and, therefore, the ability to optimally solve a programming task and acquire the necessary schemata (Robins, 2003; Rist, 2004).

The use of aptitude testing can be viewed as attempting to predict whether students will be successful programmers and comprehend the characteristics of these groups. More specifically, if I can understand the behaviour that results in better programmers, I may cultivate this behaviour explicitly. However, as evidence in the effectiveness of aptitude testing is inconclusive (Mazlack, 1980; Evans and Simkin, 1989; Hagan and Markham, 2000; Borzovs et al., 2015), the focus for understanding the conceptual difficulties of programming must be on a more cognitive view of the learning process.

2.8 Troublesome knowledge

Based on the previous section, concrete knowledge is difficult to achieve in programming education. Students must have the proper theoretical understanding of a concept and implement it concretely using appropriate strategies (Robins et al., 2003). Nevertheless, it has been reported that novice programmers usually have a superficial, context-specific understanding of programming and struggle with transferring their knowledge (Lahtinen et al., 2005). Learning to write code involves acquiring abstract knowledge (stating how a concept works) and an appropriate strategy for its application (using the concept appropriately in a programme) (Davies, 1993; Robins et al., 2003). Between knowledge and strategy, knowledge receives the most attention in research and programming

courses, which usually focus on communicating knowledge about a specific programming language.

Some of the difficulties mentioned may be due to simple misunderstandings, learning styles or motivation for learning, and others on external factors on the learners which impact their ability to understand. Moreover, some students appear to understand initially but later fail when they apply the knowledge and require additional support. To address these issues, I must try to comprehend the difficulties students face in learning programming through a learning theory.

Troublesome knowledge was initially identified by Perkins (1999), discussing challenges that constructivists must face. Perkins undertakes a constructivist approach to learning, confirming that learners take an active role in their learning. The latter requires providing activities to help the students create knowledge by themselves. Perkins classifies four types of troublesome knowledge: inert, ritual, conceptually difficult and foreign. Inert knowledge is the information known but rarely used. Ritual knowledge requires meaning and tends to be part of a routine. Conceptually difficult knowledge includes concepts that require multiple pieces of information and might be counterintuitive to students' accustomed way of thinking. Foreign knowledge comes from a different perspective than our own.

Other types of knowledge can also be troublesome. For example, Meyer and Land (2003) stated that much of the knowledge acquired by experts in a field is tacit. Tacit knowledge has become so ingrained in the experts that they use it without thinking about it, which is difficult to explain. Therefore, concepts and programming structures apparent to programming experts may not be as clear to programming novices. Generally, the concept of troublesome knowledge suggests that we should engage students actively, allowing them to create knowledge with appropriate teaching methods and measure the variables within these methods.

Threshold concepts is an educational theory regarding particularly important forms of knowledge where they are described as portals that open up new and previously

inaccessible ways of thinking (Meyer and Land, 2003, 2006). A more practical definition by Davies (2006) is that threshold concepts represent potential obstacles in the path of growing understanding. However, when mastered, they empower students to view problems in entirely new ways and think and practice like experts of a particular discipline (Davies and Mangan, 2008). An initial survey of thirty-six teachers identified several candidate threshold concepts in introductory programming courses: levels of abstraction, pointers, object orientation, instances, recursion, induction, and polymorphism (Shinners-Kennedy and Fincher, 2013).

Despite their appeal for competitive programming, threshold concepts have only a recent impact, and there are two rising issues. First, while the threshold concepts framework is highly appealing on a theoretical level, very few researchers have attempted to measure their acquisition empirically (Shanahan et al., 2006; Walker, 2013). The relatively limited empirical research has used questionnaire methods (Shanahan et al. 2006), oral recordings (Scheja et al., 2010), key reflections (McLean, 2009) and framing exercises (Davies and Mangan, 2008). Second, despite their intuitive nature, I must consider how they are identified and the methodological implications. Atherton et al. (2008) suggest that the notion of a threshold concept is itself a threshold concept. In the following section, I will discuss how the threshold concepts have emerged, the methods used in their identification process and their empirical acquisition.

2.9 Threshold concepts and liminality

The theory of threshold concepts originated from the work of Meyer and Land (2003), evaluating attributes of student learning in Economics. Since then, the threshold concept framework has become popular and gained acceptance throughout a wide range of disciplines, including Biology (Taylor, 2008), Computer Science (Zander et al., 2008), Economics (Ashwin, 2008), Engineering (Carstensen and Bernhard, 2008), educational development (Timmermans, 2014), Geography (Slinger, 2011), Mathematics (Quinnell and Thompson, 2010), and spatial awareness (Osmond et al., 2008).

Threshold concepts are not similar to the core concepts generally considered by teachers as being necessary for a subject (Davies and Mangan, 2005, Davies, 2006). Instead, Meyer

and Land (2006) specified that threshold concepts are a subset of core concepts that, once comprehended, lead to a transformed understanding. This transformed understanding may represent how people think in a particular discipline or how they perceive or experience specific situations within that discipline (Meyer and Land, 2006). Instead, core concepts do not essentially lead to a qualitatively different view of the subject, but they may help students understand and learn (Meyer and Land, 2006).

More specifically, a threshold concept is 'akin to a portal', opening up a new and formerly inaccessible way of thinking about a concept. It represents a transformed way of understanding something without which the student cannot progress. As a result of understanding a threshold concept, there may be a transformed internal view of subject matter, or even world view (Meyer and Land, 2006, p. 3). Likewise, Perkins (2006) claimed that threshold concepts act like gateways. Once students go through the gate, they reach a new understanding of specific topics central to the discipline. This greater conceptual understanding might signify ways of thinking and practising, unique to a discipline (Meyer and Land, 2003; Davies and Mangan, 2007). Consequently, mastery of threshold concepts is vital to further progress in a discipline (Meyer and Land, 2003; O'Donnell, 2010).

The Threshold Concepts framework focuses on what the students are expected to learn and the teachers' ways of thinking and practising. Meyer and Land (2003) explained the term threshold concept as a portal that opens up a new, formerly inaccessible way of thinking. Students must incorporate all of their prior knowledge on the concept, recognise any constraints of the discipline and significantly modify their views so that they cannot be misinterpreted or disappear (Meyer and Land, 2005).

These concepts signify the gateways that the students must pass to achieve their academic goals. These gateways are the threshold concepts, explicitly identified by these five characteristics: transformative, integrative, bounded, troublesome and irreversible (Figure 8) (Meyer and Land, 2003).

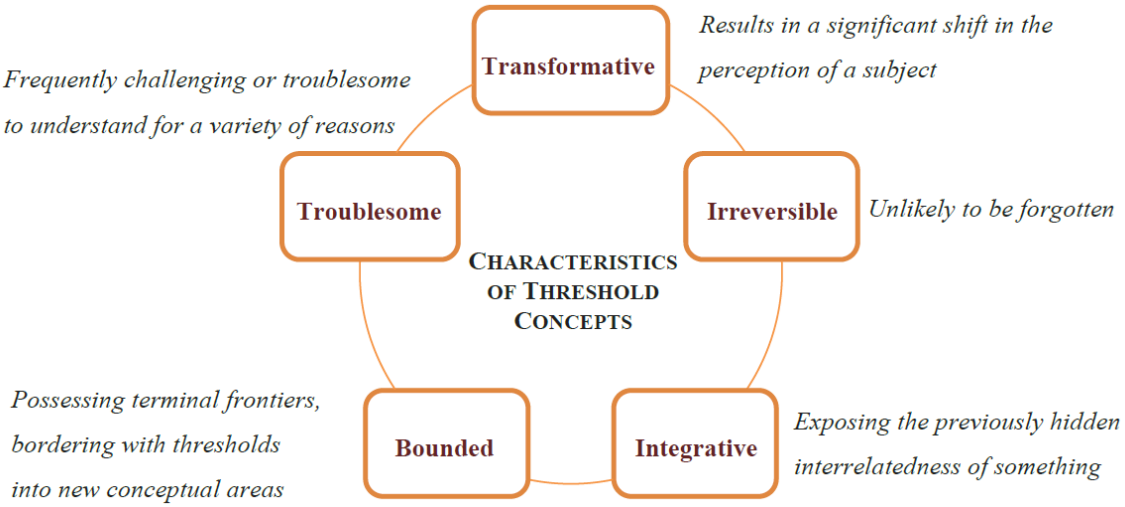


Figure 8: Five characteristics of threshold concepts (Hamm, 2016)

The two most significant are the transformative and troublesome attributes. Troublesome concepts must be perceived as far more complicated than difficult concepts. These are associated with misunderstandings, incomplete cognitive models and the failure to shift and apply knowledge from one application to another. Moreover, troublesome concepts generate confusion with current perceptions and implicit ideas (Perkins, 2006). Going through the learning process, the students ultimately grasp new ideas and acquire knowledge for the troublesome concepts of the discipline. Significantly, students obtain knowledge, but they also transform as individuals by the new knowledge acquisition (Meyer and Land, 2006).

According to their attributes, threshold concepts are challenging to understand, but they are necessary for students' progression in their disciplines. Perkins (2006) indicates that the troubling nature of threshold concepts does not derive from the actual concepts but instead from other concepts joining to create an underlying game. The underlying game improves the comprehension of the concepts at a higher level. In CS, control statements, iterations, or data structures are not complicated or transformative topics. What novice programmers find complicated and transformative is how the previous topics are used concurrently for solving a more complex task (Land et al., 2005).

Meyer and Land (2005) have defined the ways threshold concepts can be applied in both course design and curriculum development. The emphasis on the course design must be on encouraging students to overcome threshold concepts without getting stuck (Meyer and Land, 2005). When students get stuck on a concept, they often exhibit partial instead of deep learning (Laevers, 2000; Dolmans et al., 2016). Stuck students have no recollection of the course content other than the course requirements. These students do not reveal any critical thinking abilities and cannot accurately interpret the threshold concept (Meyer and Land, 2005).

In contrast, students who can conquer the threshold concept become familiar with an unconventional way of thinking that supports their abstractions and applications of the concepts. This enhancement is not only cognitive but also epistemological and ontological. Therefore, it is a permanent transformation (Meyer and Land, 2005). When students negotiate the threshold concept successfully, they claim ways of knowing and begin to think like computer scientists as they progressively obtain a new identity within a community of learning and practice (Rountree and Rountree, 2009).

The threshold concepts are essential, but not every concept in a curriculum can be identified as one. Choosing topics arbitrarily from the IOI syllabus, dynamic programming (DP) can be identified as a threshold concept. However, arrays[3], even though they are introduced in every introductory course as an essential data structure, must not be identified as a threshold concept. Arrays present new notions that require relative effort to be comprehended, but they are neither transformative nor troublesome for most students. DP is a technique for solving complex tasks by breaking the task into a group of smaller subproblems. Each subproblem can then be solved separately, and the prompted values are stored so that they do not have to be computed again. DP can be identified as a threshold concept as it encompasses all characteristics. Accordingly, when it is presented to students, they are unable to understand it and as Skiena indicated: 'Until you understand dynamic programming, it seems like magic' (2008, p. 273).

[3] A set of elements stored under the same name.

Sooner or later, as students recognise DP capabilities, they exhibit a fundamental transformation in their learning trajectories. The students can identify the base case of a DP programming task and create a recursive function that can repeatedly calculate and save the outcomes. Accordingly, students start to acknowledge that they do not have to recalculate results, and they produce source code to check and store rather than re-evaluate previously estimated results. This demonstration is an initial indication of students making magic (Skiena, 2008). The students start to apply the DP approach with other notions and in other circumstances; therefore, it is integrative. When the approach is used successfully, it is practically impossible to be unlearned; thus, DP is irreversible. DP is beneficial in the discipline of competitive programming. Simultaneously, it is an abstract concept that defines what can be calculated; thus, DP is a solid threshold concept as it has been identified through this research. I have used the concept of DP to reflect on several pedagogical inferences of threshold concepts in competitive programming. Exploring students' liminal space as they come to terms with DP will provide valuable insight into what constitutes an effective novice programmer.

A significant interpretation arising from the threshold concepts' theory is the liminal space. The notion of liminality was adopted from Turner (1969) from the Latin word 'limen', which translates into threshold or boundary. In his ethnographic studies on social rituals, namely rites of passage, he relates to adolescents' initiation into adulthood. Turner suggested that liminality helps to categorise the intermediate states of an individual and understand the reactions to liminal experiences by the way the identity is formed and the merge of thought and experience (Turner, 1969). Turner conceptualised the notion of liminality as both an inter-structural situation, betwixt and between identified roles, as well as an identity limbo, in which individuals are suspended in social space. Within an educational setting, the liminal space can be perceived as the inability of students to achieve the transformed state where they will acquire the new knowledge, and so they become stuck. The students' ability to cross the liminal space can be investigated to measure effectiveness to deal with liminality.

Meyer and Land (2005) introduce the term liminal space for the transitional period between beginning to learn a threshold concept and fully grasping it. Sometimes students

can learn threshold concepts quickly and efficiently. However, on most occasions, they require an extended amount of time. The states of liminality are transformative and usually change an individual from one state to another. Consequently, individuals obtain new knowledge and a new identity inside the community. However, this shift is troublesome as individuals experience strong, frequently negative emotions and to be successful, the individual needs to forget the old identity. The transformation can occur over an extended period and often involves oscillation between old and new states, like adolescence and adulthood. Moreover, the liminal space may involve a form of mimicry of the new state by attempting to imitate the actions of other individuals whom they perceive as having negotiated the threshold concept successfully (Turner, 1969).

Threshold concepts bring theoretical complexity but also methodological challenges. A significant issue in the identification process of threshold concepts is understanding what a threshold concept is and what makes it so. Does a threshold concept need to have all of the characteristics to be considered a threshold concept? If troublesome is the most prominent feature, what separates a threshold concept from other concepts students may have trouble with? Does the concept need to be perceived in the same manner by every student to be considered a threshold concept? If threshold concepts are defined by how knowledge is acquired and experienced by students, why are they identified mostly by teachers who have taught the concepts? Suppose any work on threshold concepts is to have real value and impact on student learning. In that case, I must initially establish what a threshold concept is, what constitutes it and the methodological implications for their identifications.

Multiple disciplines have adopted the theory of threshold concepts since their initial discovery by Meyer and Land (2003) (Taylor, 2006; Lucas and Mladenovic, 2007; Carstensen and Bernhard, 2008; Zander et al., 2008). The methodological approaches for the identification of the concepts in these disciplines included surveys, questionnaires, and reviews of previous examinations (Davies and Mangan, 2005; Holloway et al., 2009), semi-structured interviews (Zander et al., 2008; Akerlind et al., 2010) and observation of classroom behaviour (Carstensen and Bernard, 2008). Barradell argues that most of the literature concerning the identification of threshold concepts focuses on teachers' beliefs

and suggests that a consensus technique is critical to the identification process of threshold concepts (Barradell, 2012).

In Computer Science, Kallia and Sentance (2017) used a three-round Delphi approach with CS teachers to identify threshold concepts in secondary education programming until consensus and steadiness were achieved for each suggested concept. The study reported that most teachers based their suggestions on the troublesome characteristic and less on the integrative and transformative characteristics. However, the authors admit that it is unlikely that all the concepts proposed in the study are threshold concepts. However, teachers suggested that students' conceptual difficulties in programming could be explained with the theory of threshold concepts and acknowledged that recognising a threshold concept impacted their teaching (Kallia and Sentance, 2017).

Sanders and McCartney (2016) reviewed the threshold concepts identified in Computer Science and the approaches used to identify them. Rountree and Rountree (2009) suggested that the Threshold Concepts theory presents a disciplinary situated learning framework, a welcome change in perspective, away from the checkbox aspect of learning outcomes. They summarise the following proposed threshold concept examples: state, abstraction, pointers, classes/objects/instances, recursion, induction, procedural abstraction, and polymorphism (Rountree and Rountree, 2009).

A study by Khalife (2006) attempted to identify potential threshold concepts in introductory programming courses and propose solutions to support students' understanding. The author presented some universally accepted conceptual difficulties, such as lack of problem-solving strategies, and suggested that the first threshold concept students must learn is to create a simple yet concrete mental model of the computer and how it functions during programme execution. A computer model for teaching purposes is proposed, along with the results of an empirical evaluation of the model (Khalife, 2006). Boustedt et al. (2007) gathered data from both teachers (informal interviews and surveys) and students (semi-structured interviews) to discover evidence that threshold concepts exist in Computer Science. Particularly, two out of 33 concepts satisfy the criteria for threshold concepts: object-oriented programming and pointers.

In a follow-up study by Eckerdal et al. (2007), the authors used the data from the semi-structured interviews of students nearing graduation from the study of Boustedt et al. (2007). They investigated how the idea of a liminal space related to the threshold concepts identified as it provided a valuable metaphor for the concept learning process. The study supported that the students were identified as being stuck with partial or limited understanding. Moreover, all of the characteristics of liminality were present: extended time commitment, the oscillation of states, confusion, nervousness and mimicry of the new state (Eckerdal et al., 2007). Another crucial finding from this study was that students took their unique routes through liminality and got stuck multiple times at multiple places. The multiple learning paths propose that no predefined order of assignments or tasks can benefit all students. Flexibility should be considered to care for individual students' needs. No work other than the study by Eckerdal et al. (2007) has specifically addressed the liminal space in Computer Science. Moreover, while the liminal space is highly applicable to programming education, few researchers have attempted to measure threshold concept acquisition empirically by assessing students' code.

Davies (2006) argues that threshold concepts offer the means of describing the way of thinking as a distinctive characteristic of the discipline itself. Additionally, the report suggests that identifying threshold concepts is problematic due to their tacit nature as they are not often made explicit. Davies proposes two distinct methods for identifying threshold concepts within a discipline. The first method states that threshold concepts can be identified by studying how two distinct disciplines examine similar situations. The second method is by examining differences between individuals within a community of learning and those outside the community. More specifically, the distinctive approaches of novice programmers and programming experts to solve the same programming task. The second method is very suitable for teachers in any discipline to study their students. Most research for identifying threshold concepts within CS has focused on the second method by analysing students' feedback on where they became stuck while studying (Rountree and Rountree, 2009).

Land et al. (2016) suggested that threshold concepts should be perceived as an alternative to studying learners' experiences and satisfaction within a course of study. However,

identifying threshold concepts requires much more than a Likert-scale survey where students rate their difficulty level while studying each subject. Furthermore, current students are likely incapable of remembering a single standout moment in their learning, even when they can eventually understand something they previously could not (Shinners-Kennedy, 2016). They cannot mention specific details such as the circumstances or the context in which the 'learning' occurred. In situations where I deal with new learners, I can justify their difficulties based on their incomplete knowledge structures, but the outcome of the threshold identification will be inefficient. Nevertheless, the knowledge structures and meta-learning capabilities of alumni or postgraduate students are appropriate to support identifying and analysing the characteristics of threshold concepts (Shinners-Kennedy, 2016).

Knowing what constitutes a threshold concept can provide the instructor with a context in which the concept might effectively be taught. Savin-Baden (2005) proposed the notion of 'disjunction' in problem-based learning approaches. Disjunction can be identified as analogous to the notion of threshold concepts. It indicates becoming stuck in the learning process, but it can empower or restrict its effect on learning. Disjunction can be perceived as the state students find themselves in after unsuccessfully negotiating with a threshold concept. Savin-Baden points out that disjunction occurs predominantly in learning environments where problem-based learning is used because PBL requires students to assess knowledge earlier than in more traditional teaching settings. However, this is not necessarily problematic as PBL motivates students to move away from linear problem-solving methods and use more advanced strategies that require students to engage in the learning process actively. Disjunction may consequently be viewed as the troublesome state that arises when methods that require active learning such as PBL are employed, encouraging students to engage with conceptual knowledge to successfully navigate through the troublesome state (Savin-Baden, 2005).

2.10 Strategies of competitive programmers

Understanding what makes a student effective in traversing liminal space allows specific strategies and ways of thinking to be targeted. The competitive programming knowledge to deal with threshold concepts must be paired with the appropriate strategies and skills

needed to apply it. Students should understand how to use their programming knowledge to solve complex tasks. Programmers' strategies are essential in every stage of the programming process, from designing the algorithm and code-writing to evaluating and debugging the code. Code writing involves problem-solving strategies such as complete search, greedy algorithms, divide and conquer, bottom-up or top-down, which are easier to identify by the teacher. Code evaluation requires additional strategies such as code-tracing, testing, and debugging, which are not possible without using appropriate tools.

Many authors have emphasised the significance of programming strategies for successful learning outcomes (Perkins et al., 1989; Davies, 1993; Robins et al., 2003). Similar reports distinguish between expert and novice programmers using different strategies (Widowski and Eyferth, 1986). Both programming knowledge and strategies can be absent, learned but not applied, or learned but not applied appropriately, as most novice programmers tend to use insufficient problem-solving strategies (Perkins et al., 1989). Eckerdal (2009) suggests that programming concepts and practising these concepts are equally essential aspects of the learning process, and there is a mutually dependent relationship between them.

Davies's (1993) review on programming strategies argues that future research should not attempt to categorise strategies but instead focus on their origins and how they relate to their domain, associated tasks, programming language, and available tools. Moreover, research must investigate the connection between developing structured interpretations of programming knowledge and the implementation of specific strategies (Davies, 1993). Rist (2004) argues that programmers use optimal coding strategies when acquiring an appropriate schema/model. When a suitable model is not present, programmers will turn to their familiar strategies and try to create new mental models and solutions in the case of an unknown or complex task. As they become efficient programmers, students will use these strategies based on the acquired schemata/models to tackle known or unknown programming tasks. Learning technologies can monitor and assess students' programming strategies through their source code.

The acquisition of effective mental models and, therefore, successful improvement is not possible without the necessary programming strategies for accessing the knowledge and applying it to solving the programming task (Robins, 2019). Arguably, an effective programmer who utilises suitable strategies can improve by employing the correct mental models as needed. Therefore, if I can identify and exploit the strategies used by effective programmers, I can relate them with their knowledge and their acquired viable mental models and use them to instruct novice and ineffective programmers.

2.11 Mental models

Constructivism interprets student learning as creating individual knowledge structures that are constantly redefined. According to constructivist theories, learners must actively construct knowledge rather than storing it from lectures and create their unique self-constructed alternative structures (Ben-Ari, 2001). Learning something new builds upon previous knowledge in a constructivist environment, which is essential to construct new knowledge through interactions with new experiences (Howe and Berv, 2000). When students learn, they expand their knowledge structures to accommodate new knowledge, and these alternative structures occur naturally. There are instances of the application of constructivism within computer science since the development of Logo (Papert, 1980) and for the teaching of programming (Pullen, 2001; Van Gorp and Grissom, 2001).

Ben-Ari argues that we must consider two attributes to make constructivism applicable to programming education and to bring together cognitive structures with the outside world:

- A programming student has no useful model for a computer. Students can utilise a model, which is a cognitive structure, to create knowledge based on sensory activities such as working with computers and attending lectures.
- A computer is an accessible ontological reality, meaning that an accurate solution is available and effective execution necessitates the creation of a model depicting this reality (Ben-Ari, 2001).

Mental models are critical to knowledge acquisition. Models of control structures, iterations, data structures and programme design are equally essential. Ben-Ari (2001) suggests that the lack of mental models plays an important part in why students find it

challenging to learn how to programme. He argues that programmers are forced to construct their mental models from scratch with no previous models to build on. Similarly, Yehezkel et al. (2005) describe the importance of forming a mental model of a system to understand it. Wiedenbeck et al. (2004) suggested that forming mental models is a predictor for learning outcomes.

In the absence of an accurate understanding of a mental model, novices can develop peculiar theories about how programmes are executed (Du Boulay, 1986, Winslow, 1996). Mayer (1989) reported that students supplied with an accurate mental model were better at solving specific types of problems than students without the model. Derri and Pachta (2007) suggested that students must be presented with an appropriate mental model to follow, sufficient time to train with the model and meaningful feedback related to the model. Mayer (1985) suggests that specific types of mental models can be successfully taught and that such training enhances students' problem-solving skills.

One crucial mental model that programming students need to obtain is the notional machine (Du Boulay et al., 1981). A notional machine is a cognitive model of programme execution, and its goal is to provide a context for understanding the behaviour of programmes (Du Boulay et al., 1981, Mayer, 1989). In other words, a notional machine is a computer's depiction as the facilitator of a computer programme. The notional machine can be a simple description or the optimal code, and the mental model is the students' understanding of this notional machine.

Du Boulay et al. (1981) argue that we can present different characteristics of the notional machine when we use different programming languages. These characteristics can be used to identify the outcome of programme execution, and they should be supported by a reliable tool that will enable the notional machine to be observed. Sorva (2013) reviews the notional machine concept in the context of broader theoretical frameworks such as constructivism, mental models, and threshold concepts. He suggests teachers should acknowledge the notional machine as a specific learning objective and include it in their teaching process (Sorva, 2013). However, in Schulte and Bennedsen's (2006) survey, only 29% of the teachers reported that they use notional machines in their teaching practice

even though the importance of notional machines' aspects was rated highly. Consistent with fragile learning, Ma et al. (2008) reported that students with viable mental models performed significantly better than students with non-viable models.

Caspersen and Bennedsen (2007) stress the significance of worked examples as notional machines with scaffolding and feedback to support the model creation and improve learning. This review of studies points out a shift to instruction, from emphasising language syntax to developing general problem-solving skills based on a model-based approach (Caspersen and Bennedsen, 2007). Creating mental models requires conscious abstractions, assumes the confrontation with programming tasks and their optimal solutions, and propels ensuing performance for solving unknown concepts of new programming tasks (Van Merrienboer and Paas, 1990). Worked examples, as instructional instances, are accessible sources of information but the strategies used to create them are harder to make explicit. These factors underline the importance of actively creating programmes and explicitly identifying the strategies involved.

A notional machine can be a description of the code, such as a variable is like a box or a LIFO stack is like a stack of plates. They can also be representations of source code at a higher level of abstraction. Any programme is a written representation of how the programmer believes the programme is supposed to work as it contains instructions for the computer. It also contains signs and indications of the programmer's way of thinking. In the case of a good programme, the programmer has a clear concept of how the programme is supposed to work, has represented it well, and has solved the task. Conceptual clarity is a sign of a good mental model. Therefore, the programme ideally contains precise representations of these mental models.

A collection of worked examples can support the direction of independent learning activities, the creation of valid mental models, and the adaption of a supportive notional machine. From my experience in teaching competitive programming for the last decade, I have discovered that worked examples serving as reference points to a particular programming concept are invaluable to learning outcomes and student learning trajectories. An example of a segment tree design can help students create a correct

model if they are guided through the creation process. During the creation process, the students are presented simultaneously with the visualisation of the segment tree (Figure 9) and the actual code used to build it.

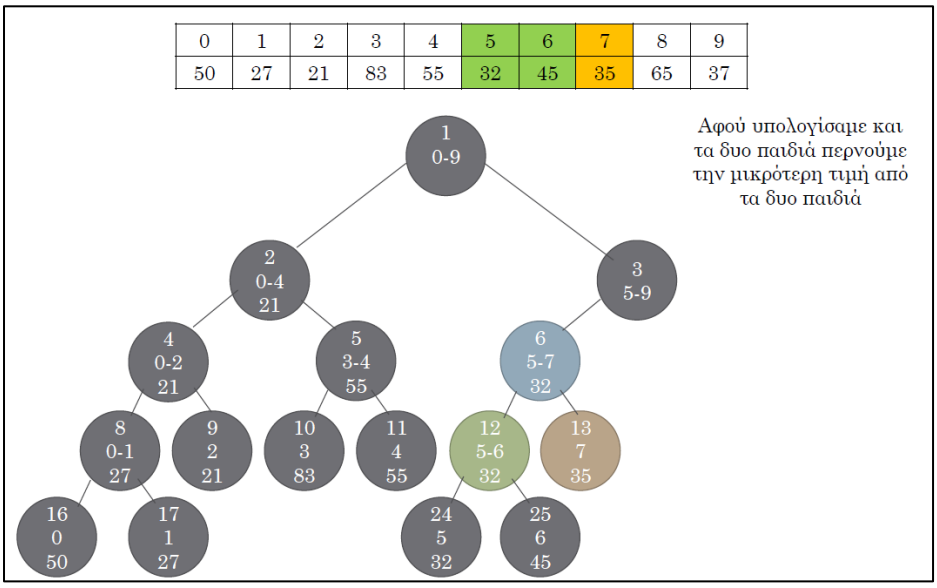


Figure 9: Visualisation of a segment tree build function

```
void build(int node, int left, int right){
     if(left==right){
          tree[node] = arr[left];
     }
     else {
        int mid = (left+right)/2;
        build(2*node, left, mid);
        build(2*node+1, mid+1, right);
        tree[node] = tree[2*node] + tree[2*node+1];
     }
}
```

Students can adapt and meet the course requirements faster and more accurately when the worked examples are sufficiently supplied before, during, and after the concepts. Based on the build function, the students are asked to create the update function of a segment tree, which is also recursive:

```
void update(int node,int left, int right, int idx, int val){
     if(left==right){
        arr[idx]+=val;
           tree[node] = val;
     }
     else {
        int mid = (left+right)/2;
        if(idx>=left && idx<=mid)
           update(2*node, left, mid, idx, val);
        else
           update(2*node+1, mid+1, right, idx, val);
     tree[node] = tree[2*node] + tree[2*node+1];
     }
}
```

From then on, the students are supported to solve an initial programming task for finding the Range Minimum Query[4] (RMQ) of an array of integers. The size of the array is initially manageable so that students can use prior knowledge (loops, arrays) to solve the problem. As the size of the array increases with a more advanced data set, the need to build and use a segment tree is made obvious, and the students comprehend the rationale and application of a crucial concept by solving a programming task as an example. The students are then assigned to find the Range Maximum Query and the Range Sum Query. I have created similar worked examples for all of the syllabus concepts.

Worked examples are an essential component of the COI framework, and they are used as a repository made available by the author for both teachers and students of the COI course. For students, the worked examples can be used as notional machines to support the creation of valid mental models and manage the learning activities. For teachers, the worked examples repository can guide the design and alignment of the instructions of the concept. The repository is dynamically reinforced throughout an academic year with the support of the learning community.

[4] A range minimum query solves the problem of finding the minimal value in a sub-array of an array of comparable objects.

2.12 Motivation for competitive programming

The students' inclination to participate and thrive in the learning procedure can be identified as student motivation (Bomia et al., 1997). Even though students might be equally motivated for solving a complex problem, the roots of their motivation could probably differ. One motivated student will attempt to solve a complex task for the satisfaction it produces upon completion, the development of knowledge it promotes, and the emotions of accomplishment it generates (Lumsden, 2004). The students' motivation is an essential aspect of learning irrespective of the discipline. Programming contests and practical programming tasks can motivate students to learn how to programme (Dagiene and Skupiene, 2004). Garcia-Mateos et al. (2009) reported that competitions could make the learning process stimulating and inspiring; thus, improving motivation.

Competitive programming is best learned by practice, and if students are to learn effectively, most of this practice must be self-motivated and self-directed. A teacher's critical role is to encourage and motivate students to engage with the learning activities appropriately. Biggs and Tang (2011) define a teacher's motivational role as making the students agree that proper task engagement is a good idea. In a programming course, a teacher has to motivate the students to improve their skills by engaging in writing programmes and participating in competitions. In competitive programming, students must be motivated to spend hours practising and competing, even when there is no assessment credit available. Therefore, an understanding of the students' pre-existing motivation is critical if the teacher wants to thrive in this.

Unfortunately, motivation is an abstract concept that is tough to measure in any meaningful way (Ball, 1977). It is possible to examine a student's behaviour to understand the type of motivation, but it is challenging to be precise. Some general types of motivation have been identified to describe why a student might value learning. Entwisle (1998) describes three generic types of motivation:

- Extrinsic - the desire to participate for attaining an expected reward
- Intrinsic - deriving from a personal interest in the subject
- Achievement - competitive, based on performing better than others.

From personal experience, students who are motivated mainly in one of these three types will have different approaches to the course. Extrinsically motivated students will not attempt a complex task without any cumulative assessment credit. Students with intrinsic motivation can be expected to learn the subject, even when there is no assessment, act more on their initiative, and form their views on the instructed material. If achievement is the primary motivator, the students will adopt the optimal strategy to achieve the best possible results in competitions.

Since the COI is a voluntary, extra-curricular course, I have identified very few extrinsically motivated students through the years. Most of the students who take the course want to learn competitive programming for the intellectual challenge of problem-solving and honing their skills, independently of any awards that may be involved. The aspiration for a future benefit for these students is perhaps in the form of a financially rewarding career in Computer Science as the most common factor, closely followed by their desire to learn. Intrinsic motivation drives deep learning, improved performance and increases with continuing engagement with assigned programming tasks.

Achievement motivation requires competitive conditions to work where students compete against each other. The ultimate achievement within the COI learning community is competing and qualifying for the national delegations. Biggs and Tang (2011) argue that achievement motivation eliminates collaborative learning as students become competitors and not collaborators. They also claim that actions are taken to gain an advantage over peers, such as key concepts are hidden or miscommunicated, hints are not disclosed, misleading advice being offered (Biggs and Tang, 2011).

From experience, in a supportive and collaborative learning environment where peer tutoring and social negotiations are cultivated from the initial introduction, these phenomena are infrequent. An instructional strategy that supports interactions among students allows them to maximise their own and each other's learning. Social negotiation is a vital component of collaborative learning as it promotes the sharing of different ideas and cooperation on problem-solving and knowledge building tasks (Duffy and Cunningham, 1996).

Effective teaching leads to improved learning which leads to increased motivation. Motivation is a direct result of quality learning. Therefore, the focus should be on designing such an effective teaching practice rather than simply motivating students. However, effective teaching is not achieved by applying general teaching rules but adapting to the teacher's potential and teaching context (Biggs and Tang, 2011). The effectiveness heavily depends on the teachers' willingness to reflect on student feedback to improve their teaching (Dunkin and Precians, 1992). Recognising and dealing with setbacks in teaching involves reflecting on the issues using a framework that offers a perspective on the teaching that helps plan interventions and make improvements. An established and efficient method for doing this is action research (Kember and Kelly, 1993), presented more extensively in chapter 3.

The key to motivation is to ensure that the teaching process is effective and that the learning activities are meaningful and valuable. This can be accomplished in problem-based learning, where meticulously designed programming tasks become the context in which students learn complex concepts and develop professional skills.

2.13 Problem-Based Learning (PBL)

Problem-Based Learning (PBL) is an established learning pedagogy that demonstrates numerous constructivist principles (Savery and Duffy, 1995; Ambrosio, 2010). PBL consists of real-life tasks attempted by students with limited support from the teacher. Explanation on how to solve the tasks is not provided, but resources are accessible to support students in dealing with the problems. Students can work cooperatively, and the teacher can help to facilitate the learning process (Kay et al., 2000). PBL tries to engage students' prior knowledge in the quest to find solutions to the tasks at hand, promote the sharing of understanding between students, clarify partial solutions, and advance self-directed learning abilities (Norman and Schmidt, 1992). Numerous reports of successful applications of PBL have been revealed in a variety of disciplines such as mathematics and sciences (Kay et al., 2000; Greening, 1999; Nuutila, 2005; Hickey et al., 2001; Simons and Klein, 2007). Moreover, new instructional methods have required a paradigm shift in how knowledge acquisition should occur in a programming course (Looi and Seyal, 2014).

Programming involves practical problem-solving and algorithm learning activities (Bennedsen and Caspersen, 2006). The subject aims to refine students' problem-solving skills (Palumbo, 1990; Apiola and Tedre, 2012) and develop their higher-order thinking skills (Ersoy and Baser, 2014). Problem-solving skills are considered one of the most crucial attributes, mainly to prepare CS graduates to enter the ICT workforce. Problem-solving skill is a cognitive process that involves exploration, analysis, and assessment (Kotovsky, 2003). Moreover, the problem-solving activity in programming requires systematic skills to synthesise a solution (Mudgett, 2014). This skill can be developed through continuous practice. Therefore, teachers must employ an appropriate teaching method to support skill development.

A student-centred approach to teaching programming using Problem-Based Learning (PBL) should be employed to allow students to deal with this new reality (Barrows and Tamblyn, 1980). PBL is a teaching strategy that is used to promote active learning. PBL originates from the theory of constructivism. Constructivism guides the teaching process with the notion that learning is a process in which the student actively constructs knowledge. Learning results from the student's actions and instruction enable and cultivate constructive activities. Instead of simply listening to lectures about the concepts, the students are presented with the assigned problem that initiates their inquiry and learning process (Hmelo and Ferrari, 1997). Learning occurs when students associate new information with prior knowledge and experiences (Duffy and Jonassen, 1992). Teachers can design meaningful problems corresponding to the course concepts and allow the students to solve them. During the process, teachers can provide guidance and organise students to discuss and collaborate. The teachers must modify their role from a knowledge-transmitter to a facilitator of knowledge. The role of the facilitator is to support and motivate students during the PBL course (Savery, 2006).

PBL provides authentic opportunities to encourage active learning, foster critical thinking, support knowledge construction, and relate the learning to real-life problems. PBL makes the problem the learning core to initiate the learning process (Bawamohiddin and Razali, 2017). The starting point for learning a concept should be a task that the students must solve (Boud, 1985). The task, or the set of tasks, is where learning starts and, in

attempting to solve these tasks, the students seek the knowledge of programming techniques and algorithms required to solve them. The disciplines do not define what is to be learned; the tasks do. However, the purpose is not only to solve those specific tasks, but in doing so, the student acquires knowledge, skills, attitudes, and know-how (Bawamohiddin and Razali, 2017). This means the tasks have to be carefully designed and have specific characteristics.

The tasks involved in PBL must be ill-structured, real-world simulated, complex, and open-ended (Duch et al., 2001; Torp and Sage, 2002; Hmelo-Silver, 2004). An empirical study by Sockalingam and Schmidt (2011) reported that the most quoted problems involved learning the outcome and generating interest. The problems were designed with an appropriate format, stimulated individual learning, and provided sufficient time for solving. Moreover, they were applicable, related to prior knowledge, and stimulated collaborative work among students (Sockalingam and Schmidt, 2011). PBL has been found to make a positive difference to the students in collaborative learning, increasing motivation, availability of peer support and problem-solving skills development (Looi and Seyal, 2014). In the study by Looi and Seyal (2014), students identified peer support as the most valuable benefit of the PBL learning environment, followed by constructing collaborative learning and learning from others.

Numerous studies support that PBL is suitable for implementation in programming courses (Kay et al., 2000; Oliveira et al., 2013). Additionally, PBL has been found to have a positive impact on students' professional careers in programming and data mining (Walker and Leary, 2009), increased motivation and reduced course dropout rate (Nuutila and Malmi, 2005), and also improved students' qualities (Cheong, 2013). For programming courses, the problems are perceived as learning essentials (Peng, 2010); therefore, the problems must have specific characteristics to initiate the learning process (Nuutila et al., 2008) Bellstrom and Kilbrink, 2009). The learning process involves a scenario related to learning outcomes, it is complex, challenging, and ill-structured, and problems require activation of prior knowledge, integration of theory and practice, and increased complexity (Nuutila et al., 2008; Fee and Holland-Minkley, 2010; Peng, 2010; Pereira et al., 2010; O'Grady, 2012).

Researchers also underlined the role and characteristics that facilitators in PBL should have. Facilitators must have problem-designing skills, develop, and distribute the tasks, deliver the concepts by intervening during facilitation, evaluate students' performance and perform reflection (Nuutila et al., 2008; Peng, 2010; O'Grady, 2012). The facilitator must also administer the learning environment (Fee and Holland-Minkley, 2010). The pedagogy and course design of a PBL approach must effectively establish a community to support student learning. In a rush to implement new technologies and online education, the notion of a learning community has often been neglected (Brodie and Gibbings, 2007).

PBL has become an integral component of programming education. However, to assign the programming tasks to students and support the learning process, I must integrate the appropriate strategies into the teaching process. Therefore, encouraging students' collaborations and providing structured scaffolding for supporting progress should be a fundamental part of every programming course.

2.14 Scaffolding and collaboration

The notion of instructional scaffolding originates from Wood, Bruner, and Ross (1976) and for producing positive learning outcomes, an appropriate social interactive framework must be provided (Bruner, 1978; Foley, 1994). Scaffolding describes the type of support offered by a teacher to support learning and, appropriately, the teacher offers guidance only with the skills that are beyond the students' capabilities. (Wood et al., 1976). It is an instructional strategy to help students with a learning task or concept beyond their competence level. As the students' understanding increases, the scaffolding is gradually removed until students are capable of solving tasks autonomously and generalising to similar situations (Foley, 1994).

Scaffolding is not limited to interactions between teachers and students. In problem-based learning environments, the notion of scaffolding is increasingly used to describe the hints and prompts from tools and learning technologies to support the learning process (Puntambeker et al., 2005). According to the literature, the integration of problem-based learning environments and scaffolding strategies increased the students' problem-solving skills (Hung et al., 2012). Resources such as compilers, online judges, and code-evaluation

platforms can themselves be used as scaffolds. Vihavainen et al. (2013) propose using an assessment system that enables the building of scaffolding into programming tasks. As the system provides some scaffolding for the programming students, it has allowed better allocation of resources (Vihavainen et al., 2013).

Dabbagh (2005) suggests a layered structured approach to scaffolding. Within a learning community, the required support level of each student may vary. For example, novice students with limited prior knowledge and effective programmers with a substantial knowledge base require a different level of support for the same programming task. Dabbagh proposes a layered structure to scaffolding to push these students to perform at their potential development zone (Dabbagh, 2003). Novice students receive the advice and support they require to confidently engage the task, while advanced students are not provided with analogous assistance. In this way, the layered approach can prevent novice students from slowing down the progress of the advanced students.

Examples of how scaffolding can be enacted in a programming course using learning technologies include:

- Online one-on-one mentoring and guidance
- Worked examples of similar or previous programming tasks
- Links to embedded cognitive tools that can reduce the complexity of the task, such as visualisation or simulation tools
- A discussion section for each task where students can seek advice on how to solve the task

In its simplest form, a collaborative strategy can be described as a teaching approach that encourages collaboration among two or more students to maximise their own and each other's understanding. From a situated cognition perspective, collaborative learning can be defined as a collection of tasks that emphasise joint construction of knowledge, joint negotiations of alternatives through debate and dependence on teachers and students as learning resources (Dabbagh, 2005). Duffy and Cunningham (1996) view social negotiation as an integral collaboration component. They suggest that in collaboration and social negotiation, the objective is to share different views and ideas and collaborate

on problem-solving and knowledge constructing activities (Duffy and Cunningham, 1996). Different groups can be formed to provide variation in social negotiations and promote peer support.

Programming tasks can engage students in online discussions to articulate their understanding of the task and make the tacit knowledge explicit by answering questions and explaining to others. In a collaborative workplace, these discussion sections can be revisited. They enable reflections on particular tasks, assessments of the learning trajectory, and performance of individual students. Peer support in evaluating another student's code can promote reflective thinking and support the understanding of threshold concepts.

Studies have found that the adoption of collaborative learning environments can improve work quality and increase the students' knowledge and programming skills (Kavitha et al., 2018). Furthermore, collaborative learning can positively impact student learning experiences (Bipp et al., 2008). A different study shows that students who use collaborative learning strategies benefit from the information and knowledge transfer process (Kavitha et al., 2017). The collaborative learning environments have been evaluated by measuring student engagement, participation, and achievement of learning goals (Dyson et al., 2003). Kavitha et al. (2018) further suggest that collaborative learning can be used as a pedagogy in programming education.

2.15 Assessment of programming code

Another critical issue that has drawn attention as a response to the issues of poor programming performance concerns how programming efficiency is assessed. Several methods exist for assessing programming efficiency, making it challenging to determine which method is the most appropriate for a specific course (Chamillard and Braun, 2000). Each assessment method has its advantages and disadvantages and is also associated with a projected learning outcome (Chamillard and Joiner 2001; Biggs, 2003). According to previous studies, the methods that are most commonly used in programming courses are written exams, assignments on computers (Chamillard and Braun, 2000; Jacobson, 2000; Chamillard and Joiner, 2001; Daly and Waldron 2004) and the use of online judges

(Cheang et al., 2003; Felter et al. 2015; Tang et al., 2016; Zhao et al., 2018; Jiang et al., 2019). Even though written exams are frequently used, many studies indicate that it is arguably not the best approach to assess a student's programming efficiency (Daly and Waldron, 2004; Rajala et al., 2016; Sheard et al., 2013).

According to Biggs (2003), the type of learning associated with written exams relates to memorisation and the skill to successfully create a written answer. The process of code-writing by hand is similar to writing an essay. The ability to structure a written answer depends on the students' ability to focus rather than their problem-solving ability. Although written exams are considered an inappropriate way of measuring programming ability among students, they can still be used to evaluate the basic understanding of programming concepts (Sheard et al., 2013).

Alternatives to written exams have introduced a computerised examination where students can use compilers to verify their solutions (Chamillard and Joiner, 2001; Haghighi et al., 2005; Rajala et al., 2016). Haghighi et al. (2005) used a computerised examination, which was considered one of the most valuable aspects of the study as it offered the students access to compiling and debugging facilities. Another study reported that students were likely more comfortable writing code on a computer with a proper editor, and coding on paper was a slow and tedious process (Rajala et al., 2016). Using a computer for the examination might considerably affect performance, as it removes the pressure of possible syntax errors. Students found testing their solutions on a computer before the final submission to be the most useful feature (Haghighi et al., 2005).

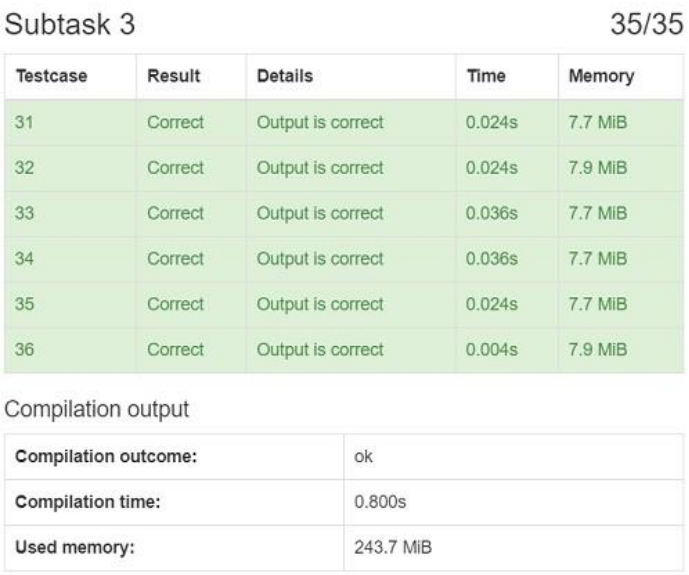
An effective tool for accessing and assessing student code is an online judge. Kurnia (2001) originally introduced the term online judge as an online platform that supports the automatic evaluation and assessment of source code submitted by users for a specific programming challenge. Most contemporary online judges have initially appeared as basic web applications introduced by universities to support programming education and students for training and participating in programming competitions. However, companies have recently used them for the recruiting/hiring processes of the world's most capable programmers and software developers.

The popularity of online judges is increasing rapidly in multiple settings and disciplines. They are systems that can provide a dependable assessment of source code submitted by the users. The source code is then compiled and verified in a homogeneous environment (Wasik et al., 2016). Their classification is vital to examine the features of each system according to its objective, whether it supports the organisation of competitive programming competitions, improving programming teaching or assisting the recruiting procedure for ICT professionals.

A crucial aspect of the design of online judges is the accuracy in measuring the execution time of programme submissions. The execution time for each test case is calculated in milliseconds. Therefore, the assessment session must be precise to differentiate minor segments of time variations and guarantee exact measurements of successive performances of identical source code for the specific test case. Numerous methods are used to estimate the execution time of submissions, such as using command-line functionalities, evaluation of hardware performing counters, code sampling and equipment. These methods offer several advantages, but some have disadvantages regarding measuring precision and accessible assimilation techniques (Ilsche et al., 2015).

The assessment process of an online judge aiming to provide a web-based evaluation of source code consists of three significant phases: submission, evaluation and scoring. In the submission phase, the source code is compiled to ensure successful execution, free of any syntax or compilation errors in the code. The evaluation phase safeguards that the memory and time constraints have not been surpassed. Moreover, the output produced by the source code complies with the specifications depicted in the task statement. Ultimately, during the scoring phase, the accumulated score is estimated based on the point distribution of each subtask and each associated test case.

The total points awarded for a specific task usually range from 0 to 100. The feedback is immediate and concise for the user submitting source code on the Michanicos platform, providing details for every specific test case as illustrated in Figure 10.



Subtask 3 35/35

Testcase	Result	Details	Time	Memory
31	Correct	Output is correct	0.024s	7.7 MiB
32	Correct	Output is correct	0.024s	7.9 MiB
33	Correct	Output is correct	0.036s	7.7 MiB
34	Correct	Output is correct	0.036s	7.7 MiB
35	Correct	Output is correct	0.024s	7.7 MiB
36	Correct	Output is correct	0.004s	7.9 MiB

Compilation output

Compilation outcome:	ok
Compilation time:	0.800s
Used memory:	243.7 MiB

Figure 10: Scoring phase of the Michanicos platform

In general, the status message of the programme execution can be each of these:

- Accepted (AC): A submission receives an AC when it executes successfully, no compilation errors exist, the output is correct according to the task description, and the time and memory limits are not exceeded. An accepted solution can receive the maximum number of points for specific subtasks.
- Wrong Answer (WA): A submission that receives a WA produced output that does not comply with the task's description or match the expected result. In simple words, for a programming task requesting the square value of an integer, for an input test case containing number 4, the associated output test case must have the value of 16. If for any reason, the submission's output for this specific test case produces a different numeric value, it will receive a WA.
- Time Limit Exceeded (TLE): A submission that receives a TLE has exceeded the maximum time limit allowed for the specific task. The time complexity of algorithms on making programmes run faster is essential, and it is a skill that students need to master.
- Memory Limit Exceeded (MLE): A submission that receives an MLE exceeds the maximum memory limit allowed for the specific task. The space complexity of algorithms on how to make programmes run more efficiently by using fewer resources is also important, and it is a skill that students need to master as well.

- Runtime Error (RE): A submission that receives a RE indicates that a runtime error has been detected during the execution of the source code.

Within the literature, there have been numerous attempts to classify online judges in distinct settings and environments (Pohl, 2006; Combefis and Wautelet, 2014; Nemeth and Laszlo, 2015; Wasik et al., 2016). Based on the results from the literature, I have considered a classification of online judges with the main emphasis on their distinctive features and their possible utilisation from the teacher/student perspective (Table 2).

Name	GUI Language	Compilers	Tasks	Users	Contests
Ideone	English	3	N/A	N/A	NO
C++ Shell	English	1	N/A	N/A	NO
HackerRank	English	3	3000	11 million	YES
HackerEarth	English	3	4000	4 million	YES
Codility	English	3	3000	N/A	NO
UVa	English	2	5000	250000	NO
Codeforces	English, Russian	3	3000	800000	YES
SPOJ	English	3	6000	60000	NO
PKU	English	2	3000	250000	YES
Timus	English	3	1000	15000	NO
USACO	English	2	150	12000	YES
Adjule	Polish	2	120	3000	YES
Hellenico	Greek	2	100	4000	YES
Mendo	English	3	200	14000	YES
Michanicos	English, Greek	3	120	270	YES

Table 2: Online judge systems surveyed. The columns include system name, the languages supported by the graphical user interface, the number of supported compilers, the number of programming tasks available, the number of users and contest availability

From an educator's perspective, the primary benefit of utilising an online judge in a programming course is the ability to evaluate students' work automatically. Initially, the source code is verified and assessed by the online judge with the maximum possible accuracy and precision. Then, possible errors caused by manual evaluation are eliminated. The teacher's responsibility is to prepare the appropriate tasks and the corresponding set of test cases originating from the problem definition. If the test cases have been adequately prepared and tested, the possibility of an incorrect solution passing all tests is negligible.

Secondly, the total time for assessing all submissions is minimal, and it allows the teacher to use the extra time to prepare additional tasks. Lastly, the feedback for the students is immediate, and it provides a detailed explanation of whether their solution is correct. There is strong evidence in the literature for the successful integration of online judges in programming courses (Cheang et al., 2003; Ala-Mutka, 2005; Ihantola et al., 2010; Fonte et al., 2013; Danic et al., 2013) and how this type of integration can substantially improve the students' learning procedure (Wang et al., 2016).

2.16 Predicting success

Given the amount of research on programming novices, it is fair to assume that we can focus on specific factors influencing success at learning programming. However, several attempts for predicting programming aptitude had limited success (Webster, 1996; Lister, 2010; Robins, 2010) as no combination of factors has been found. The most studied factors include mathematical capability and high IQ. However, these factors have produced better scores in other subjects (Pea and Kurland, 1984). Other influencing factors such as positive attitude to learning, motivation and high self-efficacy have been found to correlate with programming success (White and Sivitanides, 2002). Once again, these factors can predict success in many disciplines and not just programming. Arguably, programming students are similar to students in other STEM disciplines, and their success/failure is idiosyncratic and multifaceted. Is it possible to predict success if I focus on particular aspects of the learning process, particularly the threshold concepts of the discipline?

Meyer and Land (2005) suggest that epistemological and ontological factors affect the understanding of threshold concepts. They identify the pre-liminal variation as a 'potentially important means of opening up our understanding of why some students will productively negotiate the liminal space, and others find difficulty in doing so' (2005, p. 384). Studies reported that the pre-liminal variation has a critical role in negotiating liminality (Shanahan and Meyer, 2006; Shanahan et al., 2010). The pre-liminal variation of a threshold concept plays a significant role as it reveals the students' viewpoints and epistemological beliefs. The variation can also help distinguish why some students manage to cross liminality effectively while others fail and give up (Meyer and Land, 2006).

Meyer and Land suggest that we must go beyond the academic prerequisite checklist for investigating the pre-liminal space. We must investigate the students' epistemological beliefs in acquiring knowledge and whether they want to gain knowledge the way we want them to. Moreover, we need to check on their willingness to undergo a possibly lengthy period of confusion without any warranty that they will be successful. Teachers must not neglect but further explore students' pre-liminal variation considering the apparent effects for the ensuing students' progress or retention (Meyer and Shanahan, 2003).

Prior tacit knowledge and experience can provide knowledge segments, supporting students to engage with new concepts for required scaffolding and meaningful learning (Davies, 2006; Meyer and Land, 2006; Savin-Baden, 2008). The pre-liminal variation can be combined with students' acknowledgement of learning identity (Perkins, 2006; Savin-Baden, 2008). The learning identity is intrinsic in the sense of a student's attitudes, beliefs or dispositions towards particular contexts or experiences (Savin-Baden, 2008).

Kinchin et al. (2010) argue that determining prior knowledge is critical to understanding how students may approach a particular threshold concept as the trajectory of approach will be affected by what they already know. Therefore, identifying necessary previous knowledge for students' scaffolding learning can help them effectively inhabit, negotiate, and cross liminal spaces (Meyer and Land, 2003; Savin-Baden, 2008). The variation can also affect students' sense of knowing and self-confidence, which were found to be vital in moving through liminal spaces based on their preparedness or willingness (Savin-Baden, 2008; Shanahan and Meyer, 2006).

Understanding threshold concepts can help students obtain deeper disciplinary understanding that involves performance and signifies their distinctive ways of thinking and practicing (Meyer and Land, 2003; Perkins, 2006; Davies and Mangan, 2007). Moreover, studies have highlighted that pre-liminal variation can affect students' ways of thinking and practising during learning (Perkins, 1999; Meyer and Land, 2005; Davies, 2006). In this regard, if I want to have an accurate indication of student success, I must first identify and assess background factors, such as prior knowledge and attitudes towards learning, that are crucial for competitive programming learning.

2.17 Chapter summary

The literature review was critical for developing a framework for competitive programming education. Investigating all critical components of the COI framework has given me valuable insights for designing and implementing it. I have identified the context where I developed the project and established the theoretical background to formulate the learning environment. I have investigated the learning theory of constructivism and the Problem-Based Learning approach and how they can be applied in competitive programming. I have explored existing pedagogical models to identify the essential framework features.

The threshold concepts theory was crucial for stimulating transformations and promoting new ways of thinking. I have developed the Michanicos platform with similar or identical characteristics to those identified on equivalent systems worldwide. I have investigated methods to support the creation of mental models, optimise students' strategies, and discover threshold concepts with the involvement of the COI community. The literature review was critical in the overall progression of the study. It has been the point of reference that allowed me to see where distinctive contributions could be identified and facilitated.

CHAPTER 3: Methodology

3.1 Introduction

The research study introduces and evaluates a practical framework for competitive programming education and measures its impact on teaching and learning procedures. The framework utilises a code-evaluation platform for increasing the students' engagement and encourages innovative ways of thinking by utilising a constructivist approach. I have integrated the Michanicos platform into a competitive, problem-solving learning environment, and students became active learners through the learning process. To assess the framework's impact on students' learning and teachers' practice, selecting an appropriate methodology was critical. Such a methodology should have supported the collection of multiple forms of data to measure the impact on the teaching/learning processes and, accordingly, bring forth the ideas that the research has produced.

The case for combining mixed methods with action research is not new. Multiple studies have advocated the necessity of employing mixed methods in action research studies (Thota et al., 2012; Ivankova, 2014). Mixed methods can deliver a robust framework for action research because it produces reliable and accurate evidence. Furthermore, in a study such as this one where the empirical data and conclusions require performance improvements, there is a justification for action research to utilise effective plans for action, reflection and intervention (Lyons and DeFranco, 2010; McNiff and Whitehead, 2011; Mills, 2011). When mixed methods are applied into action research, a consistent initial appraisal of the problem can be produced, a reliable plan of action is delivered, and a thorough assessment of the process is conducted through an informed combination of multiple qualitative and quantitative data sources (Ivankova and Wingo, 2018).

The combination of action research and mixed methods can support teachers/researchers to improve their current practices by understanding a data-driven decision-making procedure (Lyons and DeFranco, 2010). Also, mixed methods can support the transferability of the action research results in distinctive contexts and settings (Ivankova, 2014).

I have used action research as a methodology for two main reasons. First, with action research, I can improve learning and enhance the teaching and learning procedures because it provides a systematic and reflective process to investigate problematic issues within my discipline (Hine, 2013). Second, I can create new knowledge and theories, generate new ideas, and justify my actions. As McNiff and Whitehead (2006) point out, a key point here is that most literature in action research reports about improving practice but much less for facilitating and increasing learning based on enhanced practice. Appropriately, this can be perceived as new theories and significant contributions to the world of ideas (McNiff and Whitehead, 2006).

I have generated theories about learning and practice to validate my claims to knowledge (Wenger, 1998). With action research, I have developed and evaluated the framework and made it available to the scrutiny of others within the IOI community. I was able to communicate with other countries what I have discovered from my experiences and what they can understand from what I have developed at the COI with the help of my learning community. I have presented the significance of my learning and invited other team leaders to learn from us and give us feedback by utilising the framework in their context and learning in their distinctive ways.

3.2 Action Research approach

To determine my research paradigm, I had to examine the sorts of data I had to gather. To evaluate the framework, I needed to analyse all of its critical elements and how they affected my practice, and the experiences of my peers and students. Initially, I identified the threshold concepts and embedded them in the programming tasks, which I made publicly accessible through the Michanicos platform. I developed the platform explicitly for this study and measured how it supported the teaching/learning processes and the students' progression. Moreover, I investigated the teaching practices and the instructional tools used in other programming courses and compared them with the framework's components. Lastly, I wanted to assess the level of improvement in students' ways of thinking, more precisely their strategies, by qualitatively studying their source code. The empirical data had to be a mixture of qualitative and quantitative information for building on their complementary strengths (Morgan, 2014). With action research, I

wanted to evaluate the framework's impact on teaching and learning, examine the outcomes and community feedback and make modifications when required as creating knowledge is a collaborative procedure (McNiff and Whitehead, 2006).

To combine qualitative and quantitative methods, I embraced the pragmatist philosophy (Peirce, 1868; James, 1907; Dewey, 1948) and adopted the pragmatic ideas of Dewey and his instrumental approach to finding meaning (Dewey, 1903). Pragmatism focuses on the research results, identifying the importance of ideas, and it is a fitting theoretical cohort for modern mixed-methods research (Johnson and Onwuegbuzie, 2004). A pragmatist researcher can reject the concept of inconsistency amongst research practices and systematically merge different techniques to meet the research objectives (Maxcy, 2003; Johnson and Onwuegbuzie, 2004). Multiple reports in the literature have instructed teachers/researchers in CS to embrace the pragmatic philosophy, to use mixed methods and to collect data from various sources such as students' performance data and feedback from both teachers and students (Clear, 2001; Creswell and Clark, 2011; Morgan, 2014). Furthermore, numerous research reports asserted that the pragmatic paradigm was favoured between researchers who studied programming ability and knowledge acquisition and individuals engaged with the teaching and learning procedures and the suitability of teaching tools (Sheard et al., 2009). Accordingly, the research objectives and the pragmatic and theoretical alignment of the study aimed to improve my knowledge, bring about change, evaluate the impact of the proposed framework and bring forth the ideas of combining the COI framework's key components.

Thota et al. (2012) used a multiple-paradigm methodology for research by combining mixed methods with action research, and I firmly believe this approach fits perfectly for my study. The study reported that the pragmatic paradigm must be adopted by researchers who want to determine how combining paradigms and employing mixed methods can be very supportive for action research in programming education (Thota et al., 2012). In another report, Mackenzie and Knipe (2006) suggested using quantitative data in supporting ways for expanding upon qualitative data. Additionally, it might be feasible to use mixed methods in every research paradigm instead of being confined to only a single method or paradigm. The issue above may undermine the strength of a

research study. I am involved in an active, dynamic and continuously evolving discipline. The ontology of pragmatism indicates that reality is constantly scrutinised, analysed, understood and, subsequently, the best approach to implement is the approach that gets the job done (Iaydjiev, 2012). The notion of pragmatism encompasses the use of a research methodology that is built on value (Johnson and Onwuegbuzie, 2004), and it focuses on resolving real-life problems (Denscombe, 2010; Feilzer, 2010). Since all of the research objectives could be met using qualitative and quantitative data, I am positive that the pragmatist paradigm was the most relevant research paradigm for this project's scope, enabling me to combine mixed methods with action research.

Although action research is typically associated with the constructivism paradigm because of qualitative methods (Stringer, 2007), the methodology merges both empirical and rational procedures that require multiple sources of evidence (Greenwood and Lewin, 2007). Thus, whenever action research requires a mixture of approaches, the pragmatic paradigm can also be considered (Thota et al., 2012). In literature and theory, the action research methodology is stereotypically situated within the qualitative research approaches. However, multiple action research reports advocated using qualitative and quantitative approaches, often combining them within the same research, comparable to the mixed methods methodology (Creswell, 2012).

Moreover, action research reports highlight the application of mixed methods in action research (Mills, 2011). There seems to be a consensus in the literature that researchers must be comfortable with quantitative and qualitative research techniques and effectively use both when engaging in action research. By using experimental and confirmatory data evaluation in a single study, I can benefit from creating and verifying knowledge (Teddlie and Tashakkori, 2009). When researchers want to assess performance, subjective interpretations and qualitative explanations, they must combine quantitative and qualitative techniques in the same research (Mills, 2011). Furthermore, the objective of action research is to generate a special kind of knowledge and to support researchers to gain an elaborated understanding of their learning environments and to be able to find answers for troublesome issues (McKernan, 1991).

Action research is a methodology that combines alternative methodologies of research that call for a significant level of researcher involvement while the research is performed. The methodology is a repetitive cycle of the following processes: planning, acting, examining and reflecting on the outcomes generated by the research (Zuber-Skerritt, 1992). Action research can be applied in educational settings to explore a particular issue and to enhance the quality of the teaching practice (Johnson, 2005). Additionally, action research offers teachers the required knowledge to reach new levels of understanding and resolve any setbacks in their practice. The objective of action research is to enhance the quality of life of teachers and their students by improving the teaching procedure (Mills, 2011).

Action researchers are considered insider researchers, and they see themselves as an integral part of the environment under investigation, often questioning collectively and individually. 'Is my work producing as I hope? How do I enhance it even further?'. A tentative action plan (Figure 13) would identify and consider issues, think of possible solutions, try them out, monitor action by collecting data, evaluate progress by making modifications, test the legitimacy of claims to knowledge and adjust practice based on the evaluation (McNiff and Whitehead, 2006).

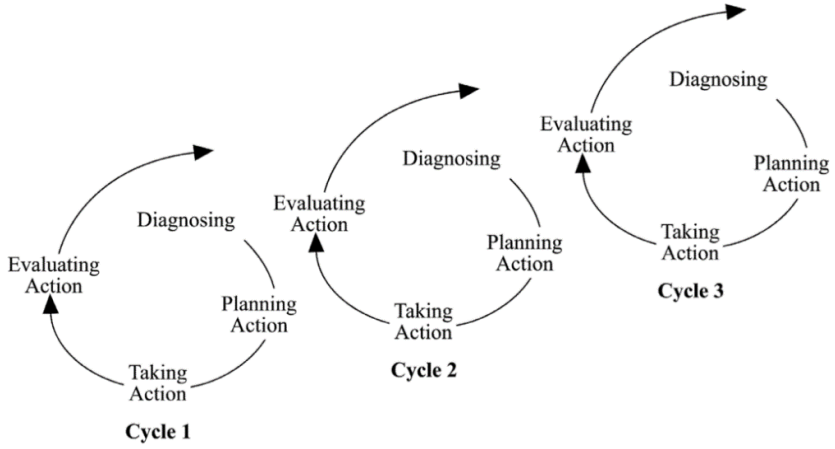


Figure 11: Action research involves continuous cycles of action and reflection (Coghlan and Brannick, 2005 p.24)

I did not wish for my work to be considered an applied theory based on the work and ideas of others but to be a knowledge creator myself. Thus, I used action research to demonstrate how I gained an understanding to improve and validate how my practice could contribute to new knowledge and new educational theories. Action research empowered me to not perceive the research process as a means of professional

development that would simply improve my students' performance. Firstly, it enabled me to demonstrate innovative practices. Secondly, it allowed me to bring forth the transforming and unique ideas within a collaborative framework. Lastly, it enabled me to offer an original contribution by presenting the theory and findings of my practice (McNiff and Whitehead, 2006). As Furlong et al. (2000) point out, teachers do not necessarily consider themselves practical theorists. To do so, I must learn how to access evidence, consider alternative courses of action and utilise such knowledge to improve practical judgement (Furlong et al., 2000). To be considered a practical theorist, I had to engage in the above processes.

Action research permits the inquiry of a troublesome issue and simultaneously enables the integration of detailed adjustments to improve the course of the research. Likewise, it provides the researcher with an increased level of flexibility which is essential when investigating an issue or a problem with incomplete or non-existent research data (Zuber-Skerritt, 1992). For this project, the literature findings of new-found frameworks or systems of training were inconclusive and primarily adapted to the requirements of other participating countries. Additionally, the identification of threshold concepts in competitive programming was absent from the literature compared to the volume of identified threshold concepts in introductory programming courses. Therefore, before the initial AR research cycle, I investigated the pre-liminal variation of students, the threshold concepts in competitive programming and certified the selected methodology. Upon the conclusion of a cycle, when the assessment reported the degree of change, I made necessary modifications before commencing the next cycle. These modifications usually involved adding new tasks on the platform, modifying existing ones, reflecting on the discussions between students and teachers and making modifications in students' learning trajectories.

Lewin (1946) defined a set of processes for an action research cycle: initial reflection, planning, action, observation, reflection. McNiff and Whitehead (2006) introduced a slightly modified version of Lewin's cycle: observe, reflect, act, evaluate, modify and proceed. In practice, this means identifying a specific concern with my practice, trying alternative approaches for getting things done, reflecting upon the results, discussing any

new understandings with colleagues, and trying another approach based on reflections. The process is cyclical, and it can be referred to as an Action-Reflection cycle. A single cycle as it was conducted for the study is illustrated below (Figure 14):

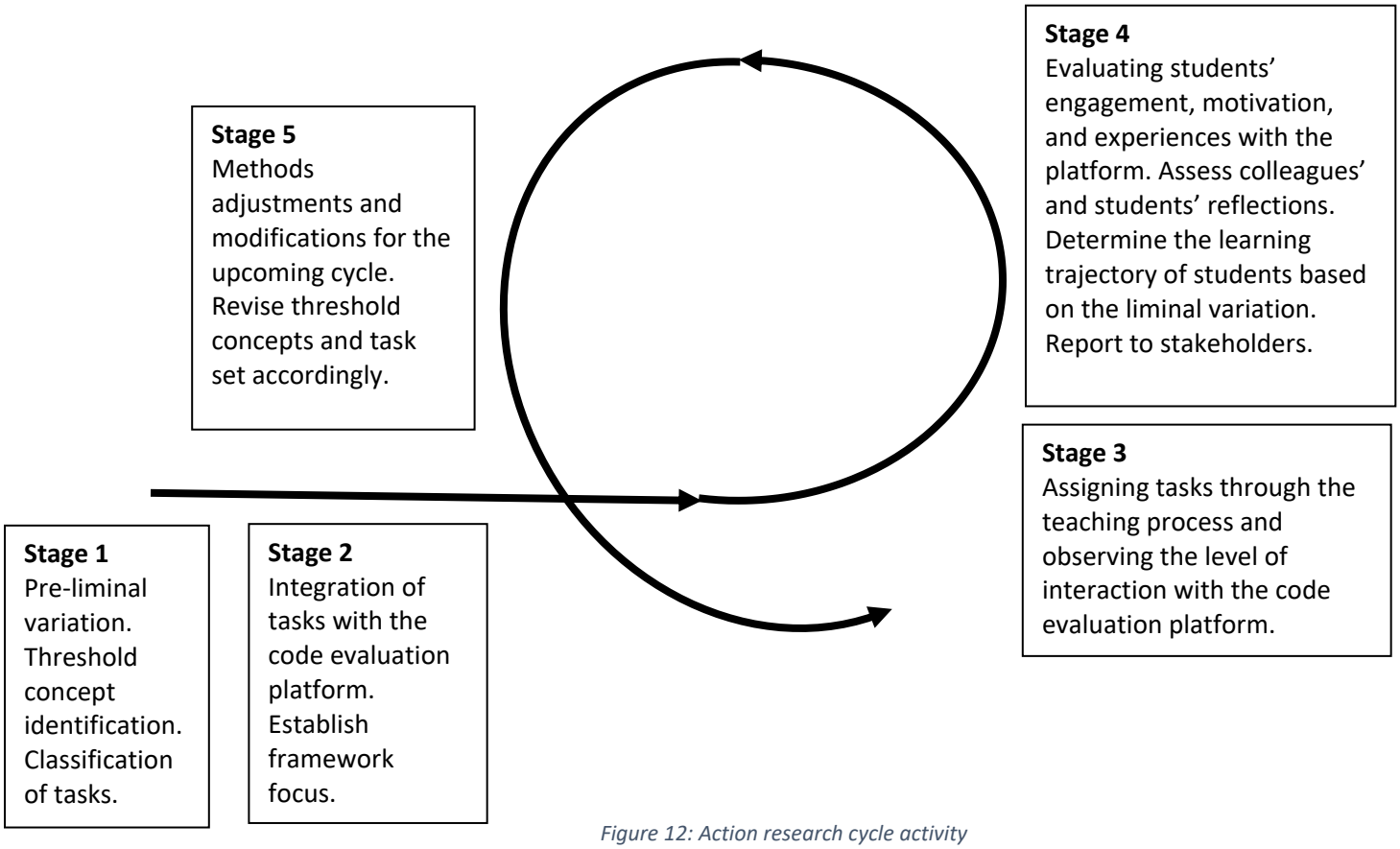


Figure 12: Action research cycle activity

As mentioned in my research proposal, for Stages 1 and 2, the pre-liminal variation, the threshold concepts detection and integration with programming tasks occurred. In Stage 3, I used the Michanicos platform to measure students' engagement with the associated material. The context of the platform included the course's modules and associated programming tasks designed to offer a stable trajectory of learning. In Stage 4, the collected performance data and the students' submissions on Michanicos served as a measurement for their progression. The scores and submissions on the platform enabled the teachers to indicate the liminal variation and how students negotiated each threshold concept. When assessing the performance data, there were no pass-fail criteria but rather a comparison of earlier performance data with the associated tasks. For Stage 5, the

necessary modifications occurred, increasingly I added more tasks embedded with identified threshold concepts, and the upcoming AR cycle followed.

In my study, an action learning set brought together the COI community and acted as a support group for the entire research project. I conducted four full action research cycles, and before each cycle, there was an analysis session of the learning set. Stakeholder representatives, COI teachers, alumni and students met and reflected on the teaching and learning processes, assessed the intermediate results and provided feedback on the teachers'/students' experience with the learning environment. Some of these sessions took place before and after the delegations had been selected. I separated the action research cycles from the year-round competition process. However, several action learning set sessions were completed during the selection process, as this was imperative for making the necessary adjustments through interventions.

The main goal of the set was to ensure that the learning produced through the framework was as authentic as possible, and reflection was the key. The students' feedback was necessary to recognise their engagement with the framework. I used the feedback from teachers for modifications and improvements on the framework and the course. Before the start of the upcoming cycle, I considered the suggestions of the learning community, and the required adjustments occurred. Students and teachers offered additional feedback via the Slack social workplace (Slack, 2019) with discussion sections for each competition round and assigned task. Students could communicate publicly through the channels created according to their programming level or privately with educators. The latter introduced the elaborated social component of the platform and proved the importance of collaborative learning and continuous quality feedback during this research study.

Educational tools and innovations have empowered teachers to accumulate and distribute an unprecedented amount of course-related data in the past decade. However, the amount of data collected does not automatically ensure that researchers use it efficiently to design or improve educational programmes and procedures. Action research provides a productive methodology that guarantees that critical information will be used to implement data-based interferences for constant academic development. Furthermore,

action research enables researchers to dismiss the role of data collectors to take on the role of facilitators of critical educational change (Hansen and Borden, 2006).

A comprehensive evaluation of the framework was essential to determine if it could accomplish its proposed educational outcomes for competitive programming education with the following issues taken under consideration:

1. Framework scope

During the initial phase of the research, COI teachers, alumni and stakeholder representatives met to define the desired outcomes of the framework and the practice. The energetic involvement of the stakeholders was crucial for defining attainable goals that would have a considerable impact on future educational changes. Furthermore, it was also crucial to receive feedback from the alumni as former course participants.

2. Data collection

I used quantitative and qualitative methods to gain a clear understanding of the effects of the framework. To respond to the study's research objectives, I needed to properly interpret the pre-liminal, liminal, and post-liminal variation and the threshold concepts involved. I collected qualitative data by using interviews and discussions with colleagues and students. Stringer (2007) provides a detailed introduction for conducting interviews, specifically on the type of questions used in action research. I found valuable the type of interviews that Stringer suggests undertaking in qualitative and action research. Moreover, the type of questions and methods needed to make the most of the information and the record-keeping process of an interview simplify the data analysis (Stringer, 2007).

I collected quantitative data using online and on-site questionnaires. Questionnaires are data collection tools that enable the researcher to collect specific data from participants, such as mindsets or knowledge (Taylor-Powell, 1998). What is crucial in questionnaire design is determining its purpose and how the information will be used to answer the research objectives. One of my initial concerns for questionnaire design was to include open-ended rather than closed-ended questions. I have used open-ended questions to gather qualitative data, enabling the participants to answer more elaboratively. However, this type of questions was more time-consuming to analyse.

3. Data analysis

It is unusual to discover how researchers analysed their data in most published research. Kirk and Miller (1986) claim that validity in qualitative research is based on what the researchers think they see in the data, indicating signs that determine how they are interpreted. Qualitative analysis has been regarded as impression analysis based on the absence of detail on how the analysis is carried out (Kirk and Miller, 1986). By using computer software in the data evaluation process, some reports claim that it can add precision to qualitative data research (Richards and Richards, 1991).

The quantitative research analysis identified the correlation between the data from a 5-point Likert scale on students' perceived coding efficiency on threshold concepts and the students' average scores from the associated programming tasks on the platform. I have used Kendall's rank correlation (Kendall, 1955) to find the strength of the relationship between the two variables. This type of correlation estimates the level of similarity between two sets of ranks, and it is more suitable to use with discrete data (Kendall, 1955). The students' perceived and actual coding efficiency were statistically analysed using the Python programming language and the statistical package SciPy (SciPy, 2019). The SciPy package can generate descriptive statistical data such as frequency distributions.

Moreover, it can also create graphical representations and provide efficient numerical practices such as routines for numerical integration and optimisation. I installed the SciPy package on a machine with Ubuntu 18.04 and used Python 3.6.1 to run the scripts that produced the results (Section 5.4). I used quantitative analysis for assessing the pre-liminal and liminal variation, evaluating the students' interactions and the overall students' experience with the framework.

4. Results Presentation and Dissemination

I have introduced the COI framework to the IOI community through the IOI 2019 conference (Eracleous et al., 2019). I have used the results from this research project to consider future framework modifications and deal with the tendencies found in the empirical data. The COI community will continue to work towards evaluation, data gathering and analysis to monitor the framework's impact and distribute information on

how the platform addresses the scientific standards of my colleagues and the needs of our students. Consequently, the evaluation and improvement of the framework will turn into a continuing, reflective, cyclic procedure in the following years.

3.3 Action Research design

My research aimed to incorporate and evaluate a framework within my learning community at the Cyprus Olympiad in Informatics for enhancing competitive programming education. The objective was to gather data through action research primarily to measure the impact of the framework, the growth of learning and how that improved learning could influence future learning and teaching procedures. The real issue here was how to accurately measure complex cognitive phenomena such as knowledge, mental models, strategies and students' levels of motivation and engagement from simple numbers and variables.

Building on notions from Greening (1999), I considered Constructivism to support the validity of the multiple viewpoints on knowledge data since it involves a significantly more subjective view on knowledge. By gathering appropriate data, the researcher can produce the kind of evidence that will enable the results to validate their claims to knowledge (Whitehead and McNiff, 2006). Overall, the triptych: theory, empirical data and research findings, was used to meet the research objectives.

Though the research examined the integration of a practical framework for teaching competitive programming, it can potentially have broader application to every level of programming education since I addressed it both from educational and technical standpoints. Accordingly, the research measured performance data and the levels of engagement and motivation with the learning environment by integrating the Michanicos platform within the framework. Furthermore, I used action research to support the successful passage through liminality, when it was apparent that interventions to the students' learning trajectories were critical. The research design I used in the study is illustrated in Figure 13.

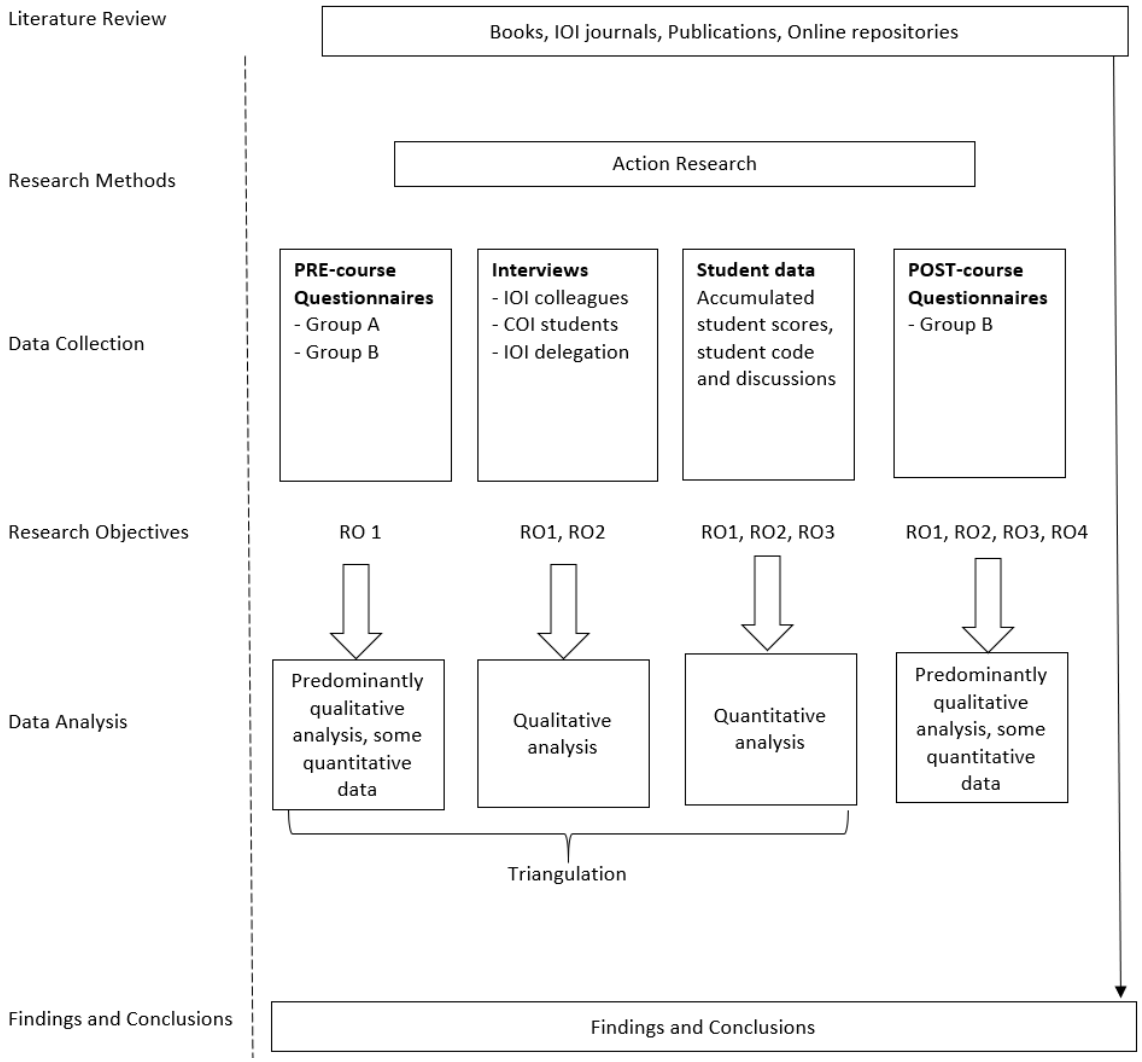
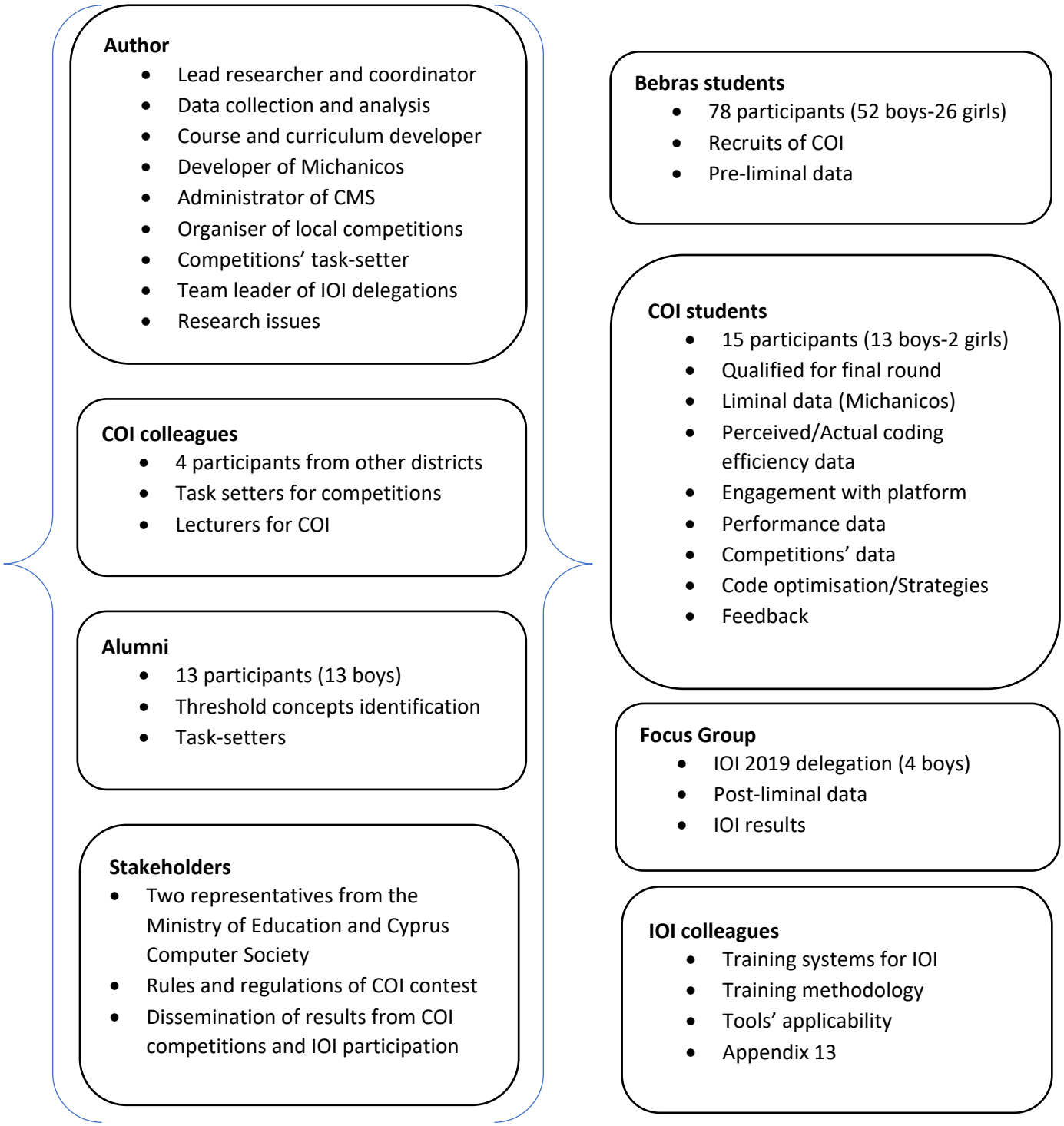


Figure 13: Research design Illustration

I scheduled the data collection at the beginning of the study because I had to use multiple data sources. The action research methodology was crucial to assess the parameters and the complexities associated with the teaching and learning processes with the support of my learning community. I have used multiple research instruments for this research study as part of the action research activity. I felt they fitted better with the corresponding methodology, so I included them here rather than in the next chapter.

The research instruments supported me in gathering the empirical data, and the following diagram presents the data provided by each group of people.



1. I used an extensive literature review on concepts such as learning theories, pedagogical models, threshold concepts, research methods and statistics, action research, mental models, strategies, conceptual difficulties, educational tools, online judges, learning programming by competitions, teaching methods, IOI tasks from the past twenty years, approaches and tools for competitive programming education, specific IOI training methods from scientific journals and IOI countries' reports.

2. I used a qualitative analysis of questionnaires (Appendix 2) before the initial action research cycle. The COI alumni have been energetically engaged with the learning community. An expert group of thirteen individuals, all above 18, consisting of past IOI contestants, provided the documentation and analysis of the threshold concepts and assessment of the liminal variation. Additionally, they clarified the students' requirements regarding preparation, feedback provision, task complexity, ways to increase motivation levels and shared their experiences with code-evaluation platforms for solving programming tasks. Predominantly qualitative information from open-ended questions was gathered (Appendix 2). I wanted to understand specific concerns, explicitly identifying and examining threshold concepts, as their views were of exceptional value and significance.

Two groups consisting of seventy-eight recruits and fifteen COI current students between 11 and 17 years of age, the participants in the competitions, were given a pre-course (Appendix 1) and a post-course questionnaire (Appendix 3), respectively. The pre-course questionnaire for the recruits was primarily focused on understanding the pre-liminal variation in the learning engagement. I investigated general information and computer literacy to identify prior knowledge and individual work ethic and initiative. I used the post-course questionnaire with current COI students to identify the liminal variation. It collected student views regarding their perceived coding efficiency and their overall experience with the learning environment using the Michanicos platform. The design of the post-course questionnaire was equivalent to the pre-course. However, I used more open-ended questions with the post-course questionnaire to receive quality feedback regarding the learning experience.

With the questionnaires, I intended to discover unique ideas of how students perceived the framework and its essential components and gather feedback on the learning environment. Additionally, I wanted to identify the distinctive ways students perceived the learning procedure. The questionnaires revealed the concerns related to student commitment, enthusiasm, performance, and setbacks in learning threshold concepts. They also focused on the interactions with the Michanicos platform, the programming tasks, the simplicity of the interface, and, lastly, the framework's limitations and potentials in general. Both questionnaires were administered on-site, and they were anonymous so that the responses would be as elaborated as possible.

3. I used a qualitative analysis of the discussions with other IOI participating countries (Appendix 13). I was concerned with gathering expert knowledge on methods of IOI training, and I wanted to investigate how these methods affected the students' programming strategies and negotiations with the threshold concepts, as they were introduced in the programming curricula of other countries. I focused on verifying the identified threshold concepts and the training methods used for IOI preparation. The support and guidance of the IOI community throughout the years had a tremendous impact on the development and establishment of the COI framework.

I used interviews with a focus group of four COI students (Appendix 4), namely the Cypriot IOI delegation of 2019, to understand why some students face reasonably fewer difficulties in understanding specific threshold concepts. I did these interviews separately from the competition process, and only a small sample of students participated. As these students interacted with the learning environment, they demonstrated increased receptiveness in engaging with the competition rounds, the programming tasks and the course context. The IOI 2019 delegation provided information on the pre-liminal, the liminal and most importantly, the post-liminal variation. I used these semi-structured interviews to identify different aspects of motivation for externalising the discrepancies in learning engagement. These students showed performance improvements and demonstrated optimal programming strategies. These improvements indicated a potentially transformative experience, and it was critical to identify how these four students negotiated liminality successfully. Therefore, I had to identify the qualitative

nuances and details of the students' effective negotiation using the selected methodology. Accordingly, the epistemological views from the students' pre-liminal and post-liminal variations were evaluated and analysed.

4. I used the action research methodology for integrating the framework into the COI course and investigating the features of the framework. With the support of my learning community acting as an action learning set, I have utilised the framework in the five districts of Cyprus, and I introduced the Michanicos platform to enhance my teaching practice. I have assigned complex programming tasks with identified threshold concepts and evaluated students' performance data throughout the COI course and local and international competitions.

I have been actively involved with a community of practitioners through IOI competitions as the team leader of Cyprus in IOI for the past five years. Moreover, I have been a member of the scientific committees of BOI 2016, BOI 2019, and JBOI 2020. I was able to explore the methods used by other countries, mainly in conferences and workshops organised during international events. By being energetically involved in a worldwide community of individuals striving to empower the next generations of computer scientists, I obtained valuable knowledge on programming education, contest organisation and technological innovations. Furthermore, my participation in the IOI community allowed me to appraise the influence of an international community on the teaching and learning processes of competitive programming globally.

Within this research, being a teacher and researcher was difficult. I was situated within the educational setting under scrutiny and, simultaneously, I was also responsible for organising and supervising the research study. I was determined to engage the research objectively and efficiently, so it was critical to differentiate between the two responsibilities. To be a productive teacher and a researcher, I have a duty to myself, my peers and my students to participate in significant educational research. I will discuss my positionality more extensively in Section 3.5.

The research findings will improve my knowledge and significantly impact what will take place in my learning community and my discipline. I have shared my research experiences to help my peers with their own research goals alongside this journey of fulfilment. I am positive that upcoming directives of educational improvements and curriculum enhancements will be affected by the ideas I have produced through the inquiries and comprehensive investigation of my research project, as well as through my practice initiatives. Engaging in educational research significantly impacted me as an individual and educator. Additionally, it demanded a considerable amount of gratification for participating in a research project that had a considerable effect on the teaching and learning procedures and the lives of the students involved.

This project has contributed to the IOI community providing a sustainable framework for competitive programming education. Furthermore, the selected methodology for this research study has enabled me to accurately measure the benefits of integrating the framework into my practice and authenticate its enhancements in the COI community. Verifiably, combining mixed methods with action research can deliver more substantial and manageable outcomes by incorporating qualitative and quantitative results to facilitate planning, implementation, assessment and adjustment (Ivankova and Wingo, 2018). Understandably, the selected research methodology has been vital for generating all of the unique ideas of this research project and successfully combining all of the framework's components into a process of action and reflection.

The project study lasted one year, the same time required for the delegations' selection process through competition rounds. Before the study, I informed the MOEC (employer and policymaker) about the research objectives, time frame, and study subjects. Additionally, I have received written permission from the CCS to comply with the MORE form, as with all of the research projects carried out under the MOEC's authority. I will present more details regarding the ethical considerations of the study in the following section.

3.4 Ethical Considerations

Ethical considerations have initially been presented in my research proposal and are of critical importance for researchers in education. Academic research is focused on the knowledge and performance of humans. Therefore, I must enforce that it will not embarrass, disturb, startle, force and undesirably influence the people participating in the research. To address this issue, I have carefully followed the ethical guidelines from the MORE form and the governing authority of the CCS for conducting this research.

Upon registration at COI, I have received initial consent from the parents of students as they have signed a document (MORE form) stating that students' scores can be exploited only for statistical analysis during the programming competitions. The competitions' scores are publicly announced for sharing the results of the qualification rounds without releasing any personal data. The CCS only discloses students' data on reports declaring the national delegations, award invitations and congratulating team members for their success.

Educational researchers confirm that educational research can help students boost their academic performances (Gbollie and Keamu, 2017; Razak et al., 2019). Hence, it must continue to be an essential aspect of social and educational practice. If I acknowledge that research can impact knowledge acquisition and, eventually, human development, it is critical to contemplate the ethical requirements to prevent interference with human rights (Tuckman and Harper, 2012). To get approval to research within the educational system of Cyprus, I have applied for permission to the Cyprus Computer Society (CCS) and the principal of Palouriotissa Lyceum (MORE form) and established specific regulations. I formally informed the CCS about my project scope, research methodology, time frame and a rigorous ethics checklist. I prepared formal consent documents with the cooperation of the CCS to notify participants and the MOEC about the research (MORE form). When I earn my degree, I must present the study's outcomes to the CCS and the MOEC.

The study involved examining the methods of teachers and opinions of students using interviews, so it was essential to maintain confidentiality and ethical guidelines. From the beginning, my peers were aware of their commitment to the project regarding the

interviews. Peers and students trusted me to investigate their methods and responses through their qualitative feedback. It was also critical to confirm that individuals acknowledged that the research was conducted anonymously so that the trust was not negotiated. I established the consent of the participants by signing the consent document using Middlesex University's ethical standards from the consent document templates (Research Ethics, 2019), the framework for research ethics (ESRC, 2015) and the MORE form. In addition, I ensured the parental written consent for interviewing students and informed the parents regarding the nature and the scope of the interviews (MORE form).

I carried out the study with thirteen alumni students that competed in IOI and BOI competitions in previous years, but they are currently not eligible to participate. The alumni filled out online questionnaires (Appendix 2). Another group of seventy-eight recruits (11-15 years of age) and fifteen current COI students, including the Cypriot delegations of IOI 2018 and IOI 2019, were also asked to complete questionnaires (Appendixes 1 and 3). I was granted parental written consent, and the deputy leader was present during the research, especially when underage students were involved.

With action research, I had to investigate within my practice by assessing the introduced framework in collaboration with my learning community. I investigated how the framework impacts the teaching and learning procedures as the COI students used the Michanicos platform to solve programming tasks associated with the identified threshold concepts. If there are ethical concerns about principles, discretion and anonymity, they can make individuals tentative to complete a questionnaire or participate in an interview. However, I have taken all the necessary precautions to establish ethical standards with my research study.

The consent documents for the alumni participants and the parents of students had to be written in Greek. The CCS established the rule as the participants were under the authority of the MOEC. I have uploaded the translated versions of the consent documents to the MORE form. Additionally, I used consent documents to obtain the students' data for the necessary travel information and acquire the parents' permission to escort them abroad

to participate in international programming competitions (Appendix 5 and 8). Each host country required an additional letter of permission (Appendix 6).

Overall, I have addressed all the primary requirements for performing ethical educational research as requested by the MORE form. Specific guidelines guaranteed no misconduct occurred, no harm was caused to participants, no absence of participant consent, no misconduct occurred, and no fraud was implicated. The initial authorisation I obtained from the CCS (Appendix 10) approving me to complete my research verifies that I addressed every required ethical standard.

3.5 Positionality in research

Positionality reflects the stance that the researcher has decided to adopt within a research study. It affects both how research is conducted, its outcomes, and conclusions. Savin-Baden and Major (2013) classify three fundamental ways a researcher can recognise and develop their positionality. First, locate themselves in the subject and acknowledge personal views that may affect the research. Second, locate themselves about the participants and consider how they view themselves, as well as how others view them. Third, locate themselves about the research context and acknowledge that the research will be affected by themselves and the research context (Baden and Major, 2013). However, no matter how reflective a researcher is, there will always be some form of subjectivity. Therefore, while exploring positionality, researchers gradually need to identify areas with potential bias and consider them (Richie et al., 2013).

For my research, I have undertaken the dual role of a teacher-researcher. I have a fundamental understanding of the epistemology of the insider action researcher (Costley et al., 2010). One mistake I wanted to avoid was considering my research role as an outsider rather than an insider, committed to the students' success in the study. It is misleading to separate a teacher's practice from the study of the action research outcomes in a setting (Herr and Anderson, 2005). When researchers authentically view themselves as insiders engaging in action research, they focus more on individual and collective change derived from actions within the educational setting (Anderson and Jones, 2000). These studies are more likely to engage in the traditional action research cycle of planning,

acting, observing, and reflecting (Lewin, 1948). The elaborated understanding of the practice and settings that result from these studies represents the outcomes of self-reflective research (Herr and Anderson, 2005).

Nevertheless, the type of thinking required for action research sits awkwardly between the intuitive decisions of the teacher and the rational and explicit analysis of the researcher. Atkinson (1994) suggests that the two parts of action research tend to work against each other. The roles of the teacher and the researcher are distinctly different and can be predisposed to be in conflict and create tensions (McNiff, 1988). McNiff (1988) suggests switching focus while engaged in a systematic and disciplined inquiry to deal with these tensions. To establish a balance between the two roles, I had to adopt the logical thinking of the teacher-researcher engaged in action research and always maintain my focus on the research objectives.

The tacit knowledge I have acquired over the past decades of teaching competitive programming raised some epistemological issues. My tacit knowledge enabled me to ask insightful questions and produce more accurate descriptions based on the understanding of the community. However, biased and impressionistic tacit knowledge, which is apparent to the insider researcher, may not be well articulated or clarified in the research and, consequently, the thesis. Moreover, as a true believer in my study and practice, I acknowledge that I have to establish procedures for dealing with bias and prejudice.

I am currently the teacher with the longest tenure in COI. My relationship with colleagues and students has been productive and supportive throughout the years. Working with other insiders has multiple benefits for a learning community with a common goal. Although this was my research from framework design to course integration and data analysis, my colleagues supported me throughout the study. Their feedback was invaluable because I was able to discuss the issues presented, consider their suggestions regarding the teaching and the research process, and use their consultation to decide the course of action.

My primary concern was conducting my research with the same students I had to prepare for the International Olympiad of Informatics. Understandably, I was concerned about possible dual role conflicts resulting in poor performance at the students' expense. At the same time, I was concerned that my focus on the students' success at IOI would lower the quality of the research. Balancing the two roles was one of the most challenging responsibilities of the study. I had to dedicate my time efficiently to review performance data, make critical adjustments for my students, colleagues, and myself, and gather qualitative and quantitative data to reach my research objectives.

Methodologically, I decided to use the IOI competition to evaluate student progress and framework effectiveness. When it comes to high-school programming standards, nothing is more complex than the tasks students have to solve in the yearly IOI competition. Therefore, my research and specifically the COI framework was not only assessed to identify whether students liked it or not. The empirical data included a significant amount of performance and contest data to measure the students' progression and the framework's impact.

As part of the research process, I have paid particular attention to my multiple tasks as an insider action researcher and a teacher to the participants and the learning community I conducted the research. I acknowledge that there may be several implications where my role and power of influence within the COI community can potentially affect the research process regarding data gathering and interpretation. I have made it my conscious and continuous effort to engage in a reflective approach with my action research set and my supervisor to develop and introduce the new COI framework. I had to ensure that my positionality could not affect the research findings and not impose my ideas on the participants. I believe that as a teacher-researcher, I have a moral directive to secure integrity. Moreover, I have an obligation to the participants who have consented to allow me to explore their views and ideas, treat them respectfully and guarantee that I appropriately disseminate the outcomes of my research.

3.6 Chapter summary

Within this chapter, I offered a justification for the selected research methodology. With the combination of action research and mixed methods being at the centre of the research design, I presented the main reasons for selecting and adopting this specific research methodology. I have used a wide range of data collection techniques and detailed data analysis to align with the pragmatic paradigm. Ethical considerations were critical in the investigation to safeguard that no unethical conduct occurred and no participant was made to feel awkward for the duration of the research process. I acknowledged my dual role as a teacher-researcher, how I recognised and dealt with tensions, and clarified my positionality in the research.

CHAPTER 4: Project Activity

4.1 Introduction

In chapter four, I present the main activities of the research project. I adopted the action research methodology to evaluate the impact of the proposed framework, facilitate the interventions used to promote student engagement, support ways of inquiring liminality and identify the distinctive programming strategies involved. The framework's learning environment was grounded on the educational theory of constructivism that provided the theoretical foundation of situated cognition. I developed the framework based on four fundamental pillars as discussed in earlier chapters: the pedagogical model of a learning community, the instructional strategy of the problem-based learning approach, the online technologies of Michanicos and CMS, and the worked examples. A productive learning environment must introduce real-life problem-based settings for learning and support knowledge construction, with special teacher assistance and peer support, within a learning community. I have evaluated all of the above features with the support of colleagues, alumni, and students.

Michanicos provided a valuable asset for implementing the constructivist approach using problem-based learning (PBL) with scaffolding to support problem-solving. Based on the literature, I anticipated that PBL would have helped students engage deeply and demonstrate much more efficient problem-solving abilities (Hmelo-Silver, 2004; Brush and Saye, 2008). Therefore, I chose the PBL approach due to its straightforwardness, composed form, and the disposition to require extended teacher supervision and feedback provision (Hmelo-Silver et al., 2007).

Two crucial components of a constructivist learning system are the teaching/learning component and the learning outcomes assessment. The continuous assessment of the acquired knowledge regarding the threshold concepts for individual students was essential to keep track of whether the learning was moving in the right direction. The COI framework incorporated both of these components to provide scaffolding in an authentic context, engage students with the complexity of the programming tasks, initiate social negotiations, and achieve a greater understanding of the concepts.

4.2 Research study activity: The COI framework design and focus

I have undertaken this research study as a teacher at Cyprus Olympiad in Informatics (COI) and the team leader of the Cypriot delegations in international programming competitions. My goal was to design and evaluate the COI competitive programming framework utilising a code-evaluation platform to improve my learning, teaching practice and students' performance. The research study was conducted cooperatively with 125 participants of the COI community, students, alumni and colleagues from the five districts of Cyprus.

I was initially concerned with several setbacks from the previous teaching methods at COI and the potential of the new framework. Students stuck in liminality demonstrated poor coding behaviours and tended to memorise large code patterns without having any interpretation of the code's rationality. Other students failed to write correct solutions, which led to their disappointment and irritation. I designed the proposed framework to support students in developing the required programming skills and improving the learning process. It was developed to encourage powerful ways of thinking, and all of the elements were thoughtfully constructed for that specific reason. The framework's components were essential, and the way I integrated them into the study determined the level of effectiveness the framework produced. I gave particular attention to the design of each component and their merging to create the learning environment. The framework's elements are presented in the table below:

Components	Procedures
Methodology	It adopted a constructivist, problem-solving, competitive environment on a code-evaluation platform. The platform was utilised as a repository for programming tasks and associated performance data for each student.
Programming tasks	The programming tasks were connected to identified threshold concepts and IOI concepts of equivalent complexity.
Computers	It promoted the personal use of a computer with installed compilers and access to the framework's components. Sample code was readily accessible to students.
Lectures	Teachers introduced a programming task, assigned the associated task set, evaluated the students' performance data and intervened when needed.

Collaboration	It encouraged interactions between teachers and students through a social workplace and the public discussions sections associated with each task.
Ways of thinking	It supported students until they reached a level of competence to solve complex tasks on their own using optimal strategies.

Table 3: Framework outline for competitive programming education

I divided the research project activity into three sections: the framework's development and integration, the course material and the programming tasks development/distribution and the assessment of the students' performance and experience. The research evaluated the performance data, competitions results, and students' feedback, behaviour and learning involved. The following is a justification for designing the framework based on the context and the limiting issues of the previous teaching methods and provides a rationale for action research.

Based on the action learning set, one troublesome concern with previous teaching approaches was the constrained association of computer usage and weekly lectures, maintaining practice and theory divided into distinct procedures. By integrating the Michanicos platform, the framework's most vital component into the teaching process, theory and practice merged. The lectures' notes were accessible within the platform and provided the scientific background for the identified threshold concepts and support for negotiating with the associated tasks. The merging led to establishing a distinct learning trajectory for all students, certifying that the post-liminal space (Meyer and Land, 2003) was within reach.

Another requirement of the proposed framework was a determined personal effort as relying solely on talent was inadequate. Talented students have been unsuccessful in former Olympiads as they have exhibited effortlessness throughout their training. I designed Michanicos to stimulate students to solve problems and accumulate points with code writing and promote self-development as students were ranked accordingly. Many programming tools were accessible through the internet (Malmi and Helminen, 2010; Cutts et al., 2011). However, none was appropriate for the competitive programming teaching and learning processes since they would arguably fail to engage and motivate students over an extended period.

An additional crucial component of the framework was the Contest Management System (CMS, 2019), a distributed system for organising and administering programming contests. The CMS promoted knowledge acquisition via the organisation of programming contests by establishing learning objectives for all the participants. The programming tasks used in the competitions were associated with the identified threshold concepts and assessed the innovative abilities of the contestants. The tasks were complex and unknown and required an optimisation of known algorithms and approaches to be solved optimally. Each task's complexity was defined by its time and memory constraints and the quality of the associated test cases, so these were addressed appropriately.

I used flexible time constraints and trivial test cases to reduce the complexity and make an IOI task solvable for novice and inexperienced programmers. However, a partial solution attained only a small percentage of the total points available. Contrary to the optimal solution, a partial solution only solved the trivial test cases for the manageable subtasks and generated incomplete results for the remaining subtasks. Optimising code and, consequently, improving ways of thinking are essential requirements of the framework. The students' determination to complete partial solutions, and increase their performance scores, establishes the degree of self-development and defines the direction of their learning trajectories.

Another challenge the COI instructors had to face was the assessment of source code. The process of assessing the students' programmes was time-consuming for COI educators in the past as it was done manually on a personal computer. To resolve the issue above, I have considered including online judges in the training procedure (Combefis and le Clement de Saint-Marcq, 2012). The advantages that online judges offer are the automatic examination and assessment of source code and providing feedback to the users. Irrespective of the vast availability of online judges, multiple concerns were raised by the COI colleagues about the scope, usability, and appropriateness of the included programming tasks. Developing the code-evaluation platform with a suitable set of programming tasks and the capability to deliver real-time assessment and feedback was a challenging project. However, the Michanicos platform made students' learning experience much more meaningful. The platform supported the teachers and

administrators in their daily responsibilities, such as creating and assigning tasks, assessing learning trajectories and cooperating with students. Consequently, the framework has reportedly supported both the teaching and the learning processes for the COI community.

To summarise, the framework's integration, intended to improve the following areas of the teaching and learning processes and the students' accomplishments were used as means of measurement:

- Students' engagement: Students' level of engagement with programming concepts can be measured and allows students to demonstrate their skills more efficiently. For defining the students' achievements and their degree of engagement, the teachers have to assess the frequency, the time frame and the accuracy of the submissions for the assigned tasks sets.
- Students' strategies: When students solve complex programming tasks, they verify what they know, and within the framework's requirements, they have to find the optimal solution with limited or no help at all. The method begins with discovering the algorithm, writing the source code and testing and adjusting the result upon submission. Students' achievements are identified when they demonstrate the application of empirical knowledge with innovative approaches for solving an unfamiliar and complicated programming task with an unknown and challenging set of test cases.
- Technology: The interaction with the technical aspects of the learning environment enables students to increase their proficiency with the usage and understanding of modern technology. Students' achievement is their ability to navigate the platform's interface and test their programmes effortlessly. Additionally, through the platform, they become accustomed to retrieving and analysing the task statement, creating a programme, identifying and correcting syntax errors and submitting their source code.
- Programming tasks: The framework contains a meticulously designed task set of distinctive difficulties embedded with the identified threshold concepts. Students' achievements are the accumulated scores for the assigned set of tasks and the individual effort for a systematic improvement of performing on every set.

- Transformations: The framework supports the students' induction into the learning community and initiates their transformations. Most former COI students will use their knowledge and abilities to solve real-life problems, engage in scientific research, achieve individual and academic advancements and flourish as new computer scientists.

4.3 Research study activity: Course context and format

I created the COI course design, which involved weekly lectures in computer laboratories, programming tasks assignments, interactions with the Michanicos platform and the social negotiations with the Slack social platform. I separated the lectures into three phases, and at the end of each phase, there was a programming competition. The first and second competition rounds had a qualifying purpose, while the third competition determined Cyprus' delegations for international competitions.

For the requirements of this research study and for effectively integrating the framework into the COI course, I organised the syllabus for 36 weeks between September 2018 and May 2019. The curricula included concepts regularly contested in IOI and the threshold concepts that I have identified in the study. The course was not an introduction to programming. Some prior programming knowledge was required as most topics are not included in any local high school curriculum. However, I recruited several students with no prior programming knowledge. They were assigned a slightly easier task set and received additional scaffolding for phase one. Each week, I taught two-hour lectures for three programming levels (juniors, sophomores, seniors), and I assigned associated tasks to students. The lectures focused on introducing a programming task, providing a worked example, and discussing and assessing potential solving strategies with students.

I used the C++ programming language for providing worked examples to students through lectures but encouraged students to use different programming languages if tasks were more manageable using Python or Java. Students were expected to solve the tasks assigned on the Michanicos platform and communicate any issues they faced through Slack with me, their teachers, or their assigned peers to promote collaborations for solving the tasks. Slack is a social workplace that enables users to communicate and share files

and other material with other students and teachers. I prepared the lecture notes, and the worked examples through an extensive literature review. I used many visualisations and code-snippets to make them more comprehensible and more natural to grasp by novice students. Each week I introduced a new topic, provided worked examples and assigned new and unknown tasks. The number of tasks associated with each topic was continually increasing, and I adjusted the task set to the programming level of individual students. Additionally, I created new tasks and added them to the platform to adjust to the needs of students who qualified and tackle possible stuck points of students who did not qualify to competitions rounds. The course material and resources are publicly available on the COI website (2019).

I used scaffolding to support the PBL approach in all phases of the curriculum. Scaffolding in the first phase was introduced at the code-implementation level, where students were given partial code and asked to complete it. This implementation-level scaffolding was essential for novice programmers because they lacked sufficient programming experience and knowledge to construct an optimal solution independently. Scaffolding in phase two was at the design level, where students were guided through the design steps for the programme solution and were required to optimise their code and improve their strategies. This design-level scaffolding was necessary for effective programmers to become expert programmers. For phase three, scaffolding was limited and was confined to discussions with peers.

The thought process I used to create the programming tasks continuously affected the students' programming capability. The process provided scaffolding for discovering good programming strategies, as students' work was constantly guided by the performed subtask classification, particularly for phase three. The programming tasks were as informative as possible. They always included sample input/output explanations or examples with expected outcomes, which provided further support for verifying the correctness of a created programme. Moreover, the Michanicos platform provided basic scaffolding with customised messages depending on the quality of the submissions. The platform provided some scaffolding for the programming tasks. It allowed better allocation of resources as teachers spent more time on more demanding scaffolding and had more

time to reflect on the outcomes. The following table shows the generalised scaffold design that I used in the course:

1. Programming Assignment: Task statement

- General information: topic, category, level, C++ worked-example
- Background: Scenario, additional info (formula, graph, visualisation)
- Task (categorised)
- Input (format and sample): Time/Memory constraints and examples
- Output (format and sample): Time/Memory constraints and examples

2. Scaffolding

A. Worked-example

Worked-example (Time/Memory complexities)

I. Task statement

- Task characteristics

II. Solution

- Spoiler
- Algorithm
- Programme structure
- Required strategy
- Optimal solution

B. Support

- Task concept (Teacher)
- Programming concept (Teacher)
- Solution description and support (Teacher, Peers)
- Programming design [Michanicos)
- Social negotiations (Slack)

For this research study, the weekly schedule below was scaffolded in the following three phases of increased difficulty. The topics assigned with each phase are based on IOI standards and represent the programming thresholds associated with different age groups. Scaffolding was gradually reduced as students progressed through the phases:

Phase One:

Topic	Duration	Tasks
Programming structures	Weeks 1-4	25-30
Basic algorithms complexity (Big O notation)	Week 5	2-4
Arrays (1D/2D)	Weeks 6-7	8-10
Basic Data Structures: Vectors, Stacks, Queues	Weeks 8-9	8-12
Sorting algorithms: $O(N^2)$ Bubble sort, Insertion sort, Quicksort	Week 10	6-8
Searching algorithms: Complete, Binary Search	Week 11	4-6
Basic string manipulation	Week 12	4-6

Phase Two:

Topic	Duration	Tasks
Functions and Recursion	Weeks 13-14	4-6
C++ STL: Maps, Sets, Pairs, Priority queues	Week 15	8-12
Graph theory (Adjacency matrix/list)	Weeks 16	4-6
Graph traversal (BFS, DFS)	Weeks 17-18	8-12
Shortest Paths (Dijkstra, Floyd-Warshall)	Week 19-20	6-8
Minimum Spanning Trees (Prim, Kruskal)	Week 21-22	4-6
Problem-solving paradigms	Week 23	4-8
Intro to dynamic programming (Fibonacci, coin change, subset-sum, LIS, Knapsack)	Weeks 24-25	10-12

Phase Three:

Topic	Duration	Tasks
Advanced graph theory (DAG, Topological Sort, Strongly Connected Components)	Weeks 26-27	2-4
Trees (Segment trees, Binary Indexed Trees)	Week 28-29	8-10
Number Theory (Prime numbers, Modulo arithmetic, Big Integer)	Weeks 30	4-6
Advanced string algorithms (Knuth Morris Pratt, Rabin-Karp, Suffix trees)	Week 31-32	6-8
Computational Geometry (Convex Hull, area of a polygon, line intersections, closest pair of points)	Week 33-34	4-6
Dynamic Programming Optimisations (Convex Hull, Divide and Conquer, Knuth)	Weeks 35-36	8-10

Table 4: Three phases of COI programming curriculum

Between each phase, I organised an on-site programming competition. The following competition format was followed to determine the final delegations:

- Preliminary round: Just before the Christmas holidays, four preliminary problems were publicly announced on the Michanicos platform, and students were granted two weeks for submitting solutions. The student scores were not accumulated to the first-round results.
- First round (four problems - three hours): The tasks were based on the topics from phase one, and students who scored 50% of the total points of the first round qualified for the second round.
- Second round (four problems - four hours): The tasks were based on the topics from phase two. Students who qualified from the first round could compete, and the top twenty students qualified for the third round.
- Third round (IOI/BOI Selection) (four problems - five hours): The tasks were based on the topics from phase three. Based on the third round results, the IOI (Azerbaijan) and the BOI (Greece) delegations were formed.
- JBOI/EJOI selection round (four problems - four hours): The selection of the JBOI/EJOI team (Slovenia) was based on the topics from phase two. Students who competed in the first round and were eligible with JBOI/EJOI age requirements could compete (up to 15 years).

I used the latest version (1.4) of the Contest Management System (Maggiolo and Mascellani, 2012; CMS, 2019) as the official contest environment for all of the competition rounds. CMS is a free and open-source grading system initially used in IOI 2012 in Italy and, except for IOI 2016 in Russia, every IOI ever since. CMS was developed to provide a scoring system that served the demands of a large-scale programming competition, with particular attention on security and adaptability.

Perhaps the most valuable quality of CMS was that it was built for the IOI community and, accordingly, several countries have used it in their local competitions. We began using the CMS (Figure 14) with our local programming competitions in 2014. This innovation has offered our delegations a critical advantage as they were getting familiar with the programming environment through the COI framework before taking part in the IOI. In a

recent development, the CMS now contains the IOI tasks from previous competitions so students can practice by selecting the contest of a specific year.

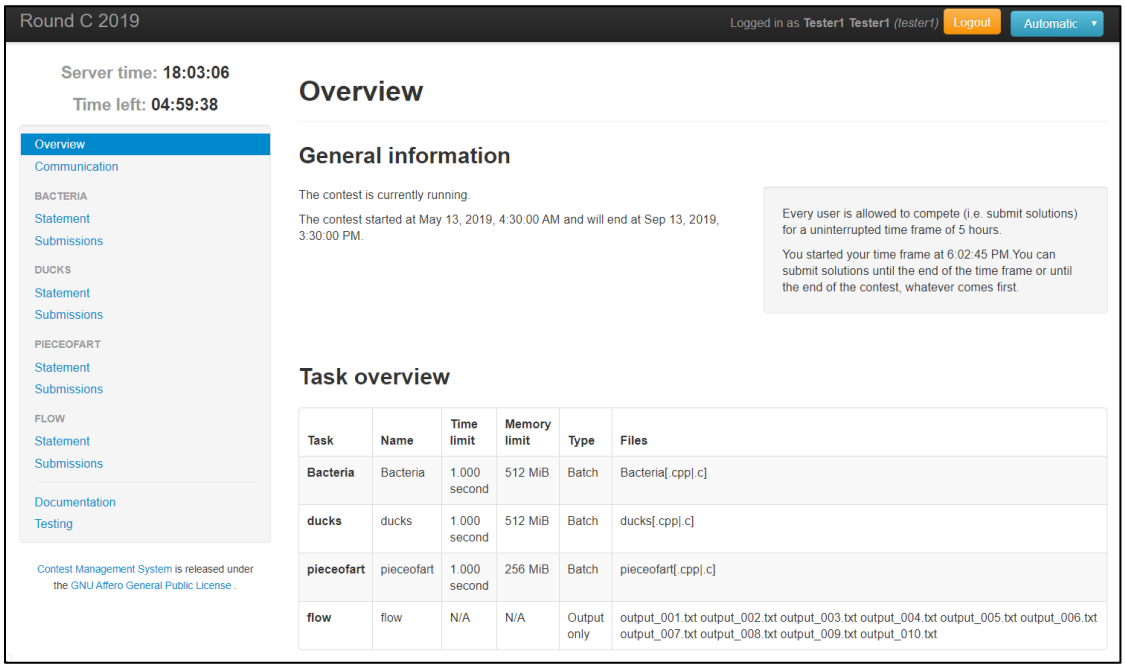


Figure 14: The interface of the Contest Management System with the third-round tasks

I completed the setup of CMS for local competitions on a single server machine that was running the services and handled the submissions. In BOI 2016, the technical committee utilised version 1.3 of CMS, and the arrangement was on three different machines. The first machine was executing only the necessary services, and the remaining two machines were processing the submissions. There were four workers (software that executes the jobs) on every machine for twelve workers. The technical specifications for the machine are as described below: processor: Intel i5-3470 3.2Ghz, main memory: 4 GB, secondary memory: 256 GB, SSD drives (128GB x2). As soon as each contest was finished, the programming tasks were migrated to Michanicos and became accessible for practice.

4.4 Research study activity: Integration of the Michanicos platform

The Michanicos platform has fulfilled an essential requirement for the COI community and its instructors, who required more effective ways of sharing and assessing tasks and running monthly programming contests. With the platform, I provided a localised online judge with an interface that supports the Greek language. It is an open-source project built upon CMS and the CMSocial engine (CMSocial, 2019). The platform's server machine

has the same technical characteristics and specifications as the CMS machine. The CMS and Michanicos are currently located in the server room at my school, Palouriotissa Lyceum (Figure 15), and I am responsible for their year-round administration, monitoring and maintenance.

Figure 15: The Michanicos platform and the CMS

The need to integrate an online judge into effective programming education originated from my desire to offer students a unique educational tool, improve the academic and technological competence of my community, and promote the ongoing success of Cypriot delegations in international programming contests. As a result, I integrated the Michanicos platform into the learning environment of the COI community in 2018.

Michanicos has established itself as an active competitive system publicly available to aspiring programmers and secondary and tertiary education teachers and students. The platform was a considerable addition to the COI. As reported (Appendix 13), not many countries participating in IOI have their own localised code-evaluation platform due to a lack of expertise or time to administer it year-round. The administration and management of the Michanicos platform are equally demanding as its development and implementation.

I set the following objectives during the platform's course integration phase:

- To improve the availability and usability of algorithmic/programming tasks and to create a vast, remote task repository. The tasks were easily accessible to the COI community, and reuse and sharing resources were encouraged.
- To offer a more significant pedagogical character to the learning environment by developing an intelligent system with real-time evaluation of tasks categorised into different tags and increasing difficulty levels.
- To decentralise the course management and allow each COI teacher to easily create, select and assign different tasks and assignments through the platform.
- To be utilised as a source code database, student scores and communication repository so that all data could be analysed and evaluated at any point.
- To offer an interactive, multilingual user interface so that other countries and non-native students could easily use it.
- To organise monthly programming competitions to challenge students, promote competitive programming learning and instruction and support the formation of a practical learning framework for all programming levels.
- To establish a community of educators and students at the local and international level for sharing experiences and knowledge.
- To examine the possibility of working with a distributed system supporting automatic evaluation so that the field of application of the code-evaluation platform can be extended to other programming courses.

For the successful utilisation of the Michanicos platform, the following areas of attention were revised continuously and were evaluated comprehensively:

1. Development and implementation of the task repository: The programming tasks of the repository were assigned to the learning objectives. The threshold concepts identified and analysed with the action research methodology were embedded and associated with specific tasks of increasing level of difficulty. Each task was explicitly tagged with the associated threshold concept and the corresponding level of complexity.

2. Improvement of the evaluation engine: The responsiveness and evaluation times of the platform were continuously evaluated as it was essential to provide students with prompt

and valuable feedback upon submission. I collected feedback from students regarding the pedagogical and technical functionalities of the platform.

3. Integration of the code-evaluation platform: Integrating the platform into the learning environment for assigning tasks and automatically evaluating and assessing the students' submissions provided numerous advantages for teachers and students. One of the most significant advantages was that I managed to identify the liminal variation for each threshold concept for each student based upon the way they interacted and negotiated with each task.

4. Platform evaluation and promotion: To determine the platform's effectiveness on the teaching/learning processes, I collected qualitative and quantitative data from the teachers/administrators and students/users. The monitoring and assessment of the platform were performed both from the teachers' and the students' perspectives. Further promotion of the platform was supported by CCS conferences, COI workshops, book publications with associated tasks on the platform and local programming competitions at all levels of education.

After two years of development and testing (2016-2018), the Michanicos platform was available online. It was introduced successfully during the Easter training camp in April 2018 at the University of Cyprus. The initial responses and reactions from the approximately forty students and colleagues actively using it during the camp were very enthusiastic and encouraging. Multiple tools and online systems have been introduced for supporting programming education (Laakso et al., 2005; Crescenzi and Nocentini, 2007; Kaila et al., 2009). Regardless of their effectiveness, all available tools and systems will produce authentic results and have genuine scientific significance if employed and utilised correctly. Several measures have improved the platform's efficiency, like the initial introduction to students, the support during the first attempts to write and submit source code, and the noticeable improvement of students' strategies. It is evident that when debating which instrument or system of training to use, it is also imperative to reflect on how to utilise it effectively to impact learning outcomes substantially.

Preliminary analysis with the original prototype Ariadne (2015) indicated that students were fascinated and welcomed the competitive aspect of an online judge in the learning process. The competitive aspect was transparent, especially when there was a leaderboard present that revealed the progress of everyone. The task development was essential for establishing student engagement with the framework.

4.5 Research study activity: Task development

Any programming competition needs to have a unique set of programming tasks to succeed. The sheer volume of tasks available through online competitions makes it challenging for teachers to discover innovative ideas for new tasks that can be both interesting and educational. The real question here is how to create appropriate tasks and what are some essential characteristics to consider when designing new tasks. Furthermore, how to deal with IOI and BOI contestants when I have to challenge arguably some of the next generation's brightest minds, considering the impact of the identified threshold concepts? Some essential characteristics of a proper IOI task were defined by Burton and Hiron (2008). However, they report that the outcome may differ depending on the target group and the objectives of the competition. Diks (2007) also suggests some similar characteristics that I have applied for the framework:

- The task statement must be unambiguous and easy to understand.
- There must be several solutions of different complexity that solve the task correctly to allow the students to use different approaches.
- The solution should not involve the reproduction of a well-known algorithm but rather a modification of that algorithm or even the combination of variations of two algorithms.
- The proposed solution should be optimal or very similar to it, and it should be extremely concise.
- For skilled students, it is suggested to have tasks that do not fall under a specified category (ad-hoc), and the appropriate algorithm is not entirely distinct.

The most challenging part of creating programming tasks is the conception of the initial idea. Inspiration for new tasks can be found in real-life problems, and even people

unfamiliar with CS can come up with interesting ideas. The real challenge here is that the author must create the solution(s) and the problem concurrently. One setback is that task creation from scratch can be very time-consuming, particularly when there are multiple solutions for the task that I must consider. However, new tasks do not have to be developed from scratch. Combining tasks to create new tasks can also be very effective (Burton and Hiron, 2008). In their report, Burton and Hiron (2008) state that the combination of ideas from previous tasks can be used together to form ideas that have not yet been explored.

Pankov (2008) supports the claim that real-life situations can be used for inspiration when creating new tasks. Real-life situations can yield original context that can be formulated elegantly and provide an equal advantage to novice and experienced student programmers. He suggested that natural sciences contain numerous fascinating laws and theorems that are highly appropriate for programming tasks. For Pankov, three possible task types can be considered if all of the characteristics are known: optimisation, combinatory and interaction. For example, Pankov demonstrates how conservation laws[5] can be used as the foundation of a programming task where the contestants need to discover the minimum possible speed of combined pointwise objects (Pankov, 2008).

Forisek (2006) presented tasks used in former IOI competitions that include alternative areas instead of standard tasks that focus on creating cost-effective algorithms. Forisek's study suggests that there is an opportunity for learning about various other qualities, and the scope can be situated on other notions than algorithms' effectiveness (Forisek, 2006). Many data structures that are used in programming tasks can be compared to situations in real life. Accordingly, students do not have to be familiarised with the principles of these structures. Graphs (Figure 16) are used extensively in programming tasks for every complexity level. They are easy to understand as they can be used for modelling cities (vertices) and roads (edges), and they are relatively trivial to represent in code. Therefore, tasks with simple graphs become accessible to younger students (Manev, 2008).

[5] Specific quantifiable properties of a physical system do not change over time, regardless of the system's evolution.

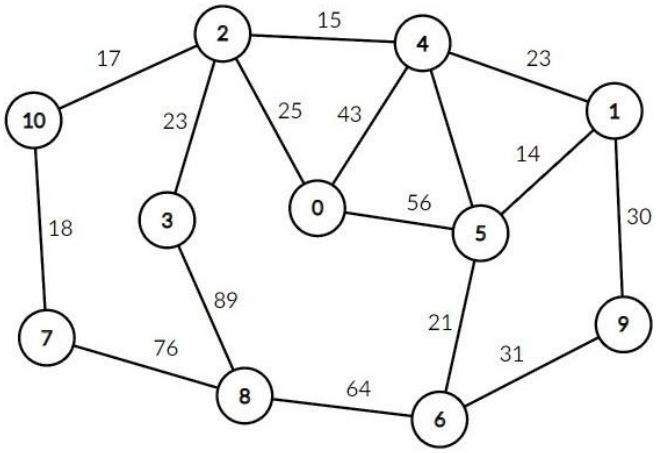


Figure 16: Example of a graph depicting cities and distances

The initial idea for a programming task is only the beginning, and much more work is required. Diks (2007) proposed best practices to be considered in the task development process for any programming contest. The main idea for the proposed practices is the rigorous application of the task development process so that the quality of the tasks is assured and possible errors are detected before the contest. When these errors occur during the contest, it would be practically impossible to correct them (Diks et al., 2007). Verhoeff (1990) has also reported detailed rules for developing a task set for a programming competition. These rules emphasised ACM (Association for Computing Machinery) programming competitions and proposed that every task should have its setter/author directly assigned to it.

For reviewing the initial task ideas, several questions must be answered: Is the task formulation comprehensive and unambiguous? If the information in the task needs to be explained extensively and explicitly, then the task is not appropriate. Is it a textbook task? If the task solution is trivial and tests only a particular algorithm or a technique instead of the students' ways of thinking, the task is not appropriate. Is the solution to the task running in polynomial time? Otherwise, it is impossible to calculate the results in a reasonable amount of time within a contested environment. How many possible solutions exist for the specific task? If multiple solutions with ranging difficulty and different time complexities exist, then it is a highly appropriate contest task. However, special attention should be given to ensure that no trivial solution to the task exists. Lastly but not least,

can the task be solved by a high school student? As the boundaries of programming education for IOI participants are extending, it is compulsory to keep in mind that the expected programming knowledge of high school students is covered in the IOI syllabus (IOI Syllabus, 2017). However, the knowledge and programming strategies involved in solving such tasks go way beyond their school programming education.

Diks (2007) also suggests that task analysis should be the most time-consuming part of task development. Therefore, several solutions should be included in the evaluation report with different approaches, at least two programming languages, including the optimal and one or two sub-optimal solutions. The task spoiler, a detailed explanation of the proposed optimal solution, is critical in analysing the evaluation report. The spoiler explains the author's approach in solving the task but by no means should it be limited to only the proposed solution.

For task development in this research study, in the case of batch tasks (input-output tasks), the COI teachers and alumni created a set of 10-50 test cases. In simple words, if the programme received the data from the input file *input1.txt*, the expected output precisely matched the contents of the output file *output1.txt*. Otherwise, the submitted solution was wrong. The objective of test cases was to differentiate between correct, partially correct and wrong solutions. The differentiation was extremely complicated as partially correct solutions could be separated mainly through time and memory constraints. Explicitly for time constraints, the test cases were set to put a strain on the asymptotic time cost and not on the actual running time of the programmes. In general, solutions that were twice as slow as the optimal solution could score 100 points. Slower or partially correct solution solutions could receive up to 50% of total points, and 100 points were awarded for efficiency. This level of meticulousness on the test cases was tough to accomplish. A quick solution was to increase the data sizes of tasks, but the computers' RAM availability and testing times were often restrictive.

A more appropriate strategy that the action learning set decided to utilise was to group the test cases into subtasks for most tasks. A submitted solution was awarded the number of points associated with a specific subtask only if it solved all the test cases in the given

subtask. Furthermore, I used subtasks to achieve optimal point distribution among contestants. The first subtasks were usually the easiest and less demanding. However, they awarded fewer points, usually less than 10%, and the last subtask was the hardest but awarded the most points, usually more than 50%. I included approximately 3-5 subtasks in every task, and each could be solved using a different strategy. For formulating a task, I combined all of its elements. The task description was a short story explaining the task in detail. Input and output formats were precise and difficult to misinterpret. Specifically, the sample output was not just a yes/no answer but a numeric value quickly determined by the sample input. For reference, consider the tasks 'Icarus' and 'Lefkaritika' that I have created for the Balkan Olympiad in Informatics in 2019 and 2016, respectively (Appendix 7 and 11).

4.6 Research study activity: Task assessment and evaluation

For publishing a task online on the platform and assigning it to students, it was necessary for the COI teachers to initially prepare all of the components discussed in Section 4.5: task statement, test cases and proposed solution. Then, I used the platform's administrator panel to upload the task statement and the test cases (with subtasks if applicable), set the time and memory limits and test the correctness of the solution by submitting my source code. Establishing the task correctness is critical, and task testing before assigning is mandatory as it determines the validity of the contest, especially in a live competition setting. In the past, I have witnessed multiple incorrect test cases, even in IOI competitions, created with wrong solutions. These mistakes were corrected during the live contest causing delays and uncertainty among contestants.

To enforce the validity of the tasks, specifically the ones used in the competitions during this study, each task published on Michanicos had a task setter and at least one task tester. Both of them were either COI instructors or alumni. The task setter was responsible for creating the task statement, the test cases, the proposed solution(s), and uploading the task on the platform. The task tester was responsible for proofreading the task, correcting any errors in the task statement, or even simplifying the terminology and vigorously testing the task with multiple solutions, preferably one for each subtask. For

each solution submitted, I verified the points and test cases awarded for each subtask. If the task tester gave the final approval, I published it and assigned it to the students.

Even when a task gets approval and is published on the platform or within a competition, another form of assessment takes place after the students have attempted to solve the task. The assessment needs to provide a thorough analysis of the task's appropriateness to serve its scientific purpose. The complexity level within a set of tasks needs to vary from one task to the next. Not all tasks can be trivial, and not all can be highly complicated. As a general unwritten rule among task setters, an optimal point distribution among contestants is preferred. Task setters try to avoid giving away too many full scores (100) and, at the same time, avoid having too many zeros on the same task. In IOI 2018, the task 'Combo' on the first day of competition had a record of 185 full scores and only seven zeros out of 336 contestants (IOI Statistics, 2019). This issue was resolved on the second day of competition with the addition of two tasks, 'Tolls' and 'Meetings', where the corresponding numbers were 1/142 and 0/60. The total full scores of the remaining tasks, not counting 'Combo', were a mere 55 (Table 5).

	Combo	Seats	Werewolf	Doll	Tolls	Meetings	Total
Average	73.83	14.08	30.48	31.19	15.34	21.56	186.48
Total 100s	185	1	17	36	1	0	240
Total 0s	7	93	87	66	142	60	455
Ratio	26.43	0.01	0.20	0.55	0.01	0.00	**0.53**

Table 5: Statistics of IOI 2018 tasks

What can be deducted from these statistics is that I must prepare more tasks in every competition than needed. Also, I should always consider reasonably straightforward and incredibly complex tasks, but these should be used cautiously and not in abundance. Thankfully, the second day's tasks can be switched in IOI competitions if a better point distribution is required. However, this is very hard to predict in competitions that run in a single day. An overall proper point distribution in an IOI competition means avoiding many ties, making the classification and the awarded medals clear. There should only be one or two maximum scores of 600 points and as few zero total points as possible. Understandably, this is the optimal strategy for the local single-day competitions.

However, this is not always the case. There is always the possibility that a meticulously prepared and tested task does not produce the expected results. The nature of competitive programming tasks and the dynamic format of the IOI curriculum continuously requires more innovative and complex tasks. As a result, the most challenging assignment is predicting how the students will perform with a given task regardless of its complexity. A reasonably simple complete-search (aka brute force) solution examining all possible outcomes should not be allowed to score more than 20% of the total points. On the other hand, no task should restrict contestants from scoring at least 20% on average total points. The ideal shape of the scores' histogram should be a reverse J-shaped of distribution. An example of a highly complicated task is illustrated in Figure 17 below:

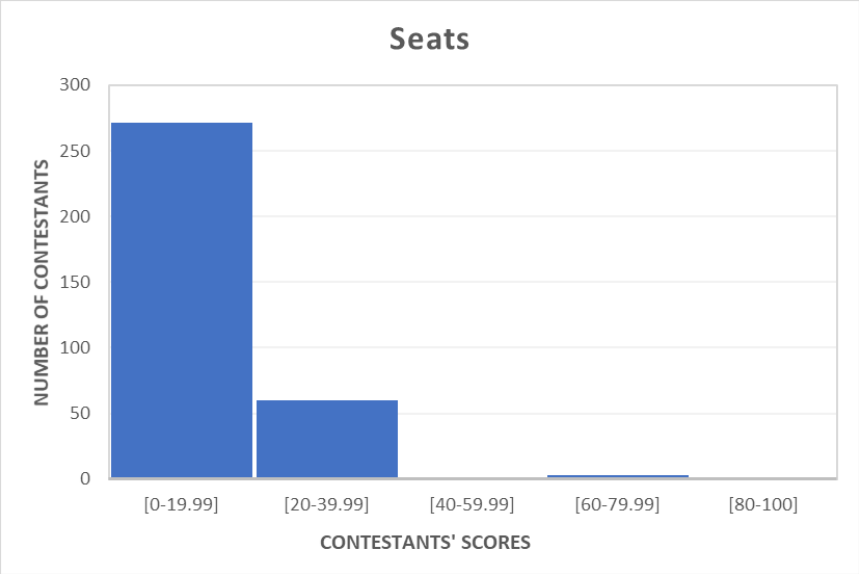


Figure 17: Task Seats point distribution (IOI 2018)

With the study's action research methodology, one measure of controlling this type of uncertainty as much as possible was an exclusive analysis session where the proposed tasks' setters and testers solved and critically assessed all of the tasks and made critical recommendations for the action learning set. The primary goal was to investigate if there was any partially correct solution that was easy to implement, notably easier than the proposed solution, which scored a relatively high number of points with the actual test cases used in the competition. Additionally, several more manageable subtasks were considered for difficult tasks to help contestants get some points even on the hardest

tasks. Collectively attempting to predict students' results for any competitive programming competition has a higher possibility of making better decisions, avoiding unfitting tasks or test cases, and ensuring the required point distribution. In the next section, the levels of interaction with the platform are discussed.

4.7 Research study activity: Michanicos interactions

The Michanicos platform has two access levels set by the action learning set. The first level is for students, and the second level is for teachers and alumni. Students can register, submit solutions, view feedback and statistics and track their submissions and scores. The teachers have all of the students' privileges, and, additionally, they can add course material to the platform, review all submissions, add/modify programming tasks and evaluate students' progress.

4.7.1 Student-level interactions

I introduced navigation through the Michanicos platform to students in the first week of lectures. The menu (Figure 18) includes the following items: Home, task archive, ranking, sign-up, login/user and language selection. The platform's interface supports both English and Greek, but other languages can be added.

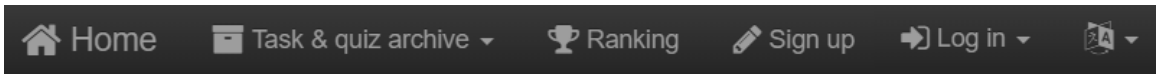


Figure 18: Michanicos platform menu

Under the task archive selection, the students can view all tasks, tasks arranged by technique/category and tasks arranged by events. Registration is not required to view the tasks' statements, but only registered users can submit a solution for a task. I asked students to register (Figure 19) on the platform and submit their solutions to a sample task during the first lecture. Each user's credentials were saved on the server and retrieved by administrators upon request for password changes or updates.

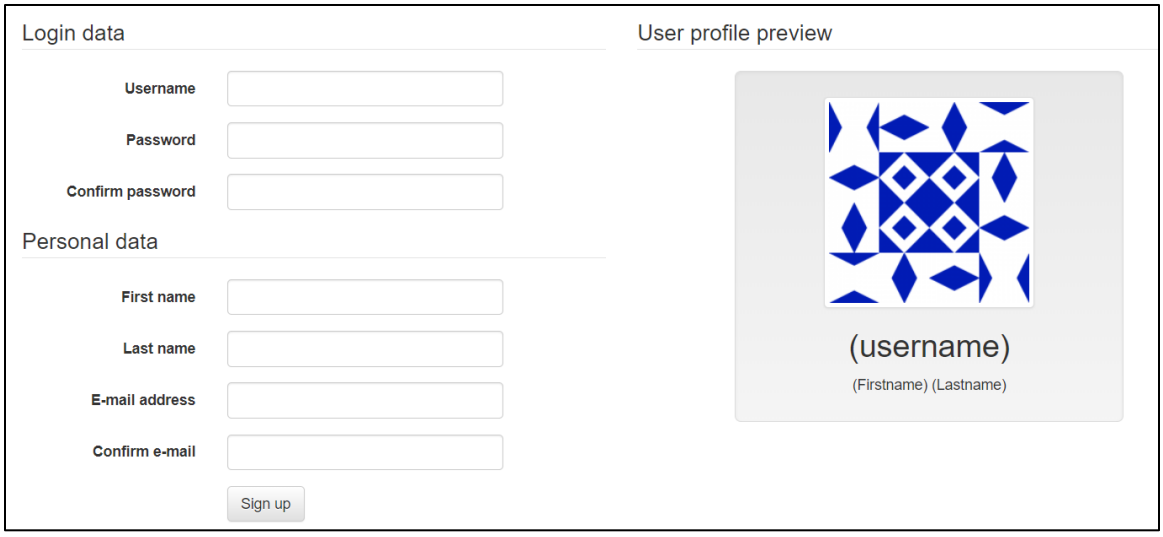


Figure 19: Michanicos registration form

I organised the tasks in specific tags. Each tag represents one particular topic from the IOI/COI syllabus (IOI Syllabus, 2017). For selecting the tasks associated with one tag, the students were required to follow the tag's URL. Tags can be created for each competition round as well. Currently, there are fifteen topic related tags and thirteen contest related tags (Figure 20). Topic related tags on the Michanicos platform include the following topics in alphabetical order: Ad-hoc, binary search, complete search, data structures, dynamic programming, geometry, graphs, greedy, implementation, MST, number theory, shortest paths, STL, strings and trees. These tags cover all of the topics that are found in IOI tasks. Some tasks can be found under two tags. General-purpose tasks that do not fall under any category are included under the ad-hoc tag.

Figure 20: Tags on Michanicos

By selecting a tag, the students are presented with all the tasks under this topic. The students are informed about how many points they have been awarded for each task they have solved (Figure 21).

Figure 21: Tasks under implementation tag

By selecting a task, students are presented with the task statement and the task menu (Figure 22). The statement is in pdf format and can be downloaded on the computer.

Figure 22: Task statement for task Wasawa

The task menu includes the following options:

Statistics: Students can view the statistics for the selected task. Statistics include the total number of users who have solved the task correctly, the total number of users who have tried to solve the problem, the total number of correct submissions and the total number of submissions. Moreover, a list of users with the fastest solutions is presented. These statistics are an initial indication of the task's complexity level (Figure 23).

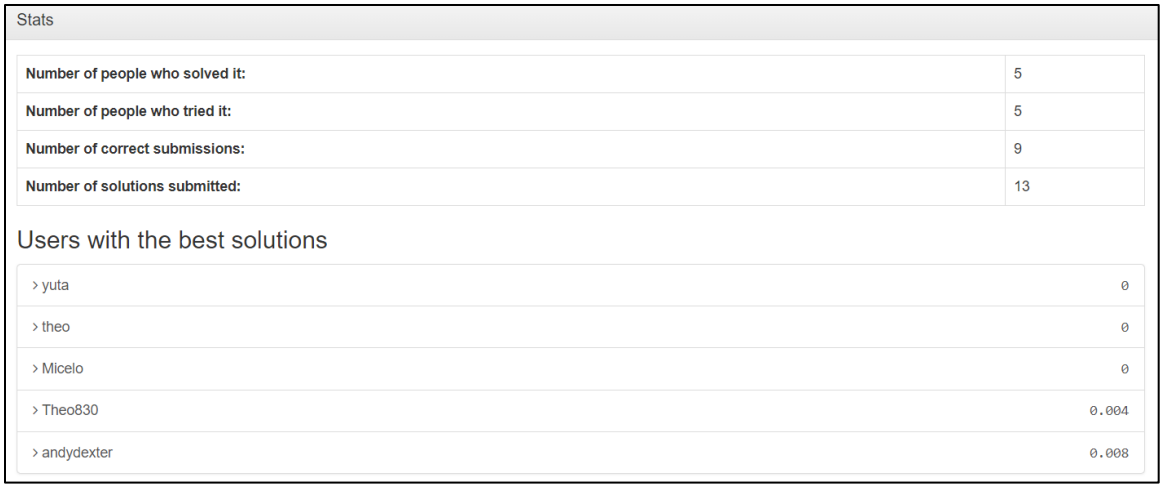


Figure 23: Statistics for a selected task

Submissions: Students can submit a solution only if they log in using their credentials. They have the option of loading a file they have previously written and tested on a local compiler, or they can use the platform's online code editor to write their source code. The code editor highlights language commands and improves code visibility (Figure 24).

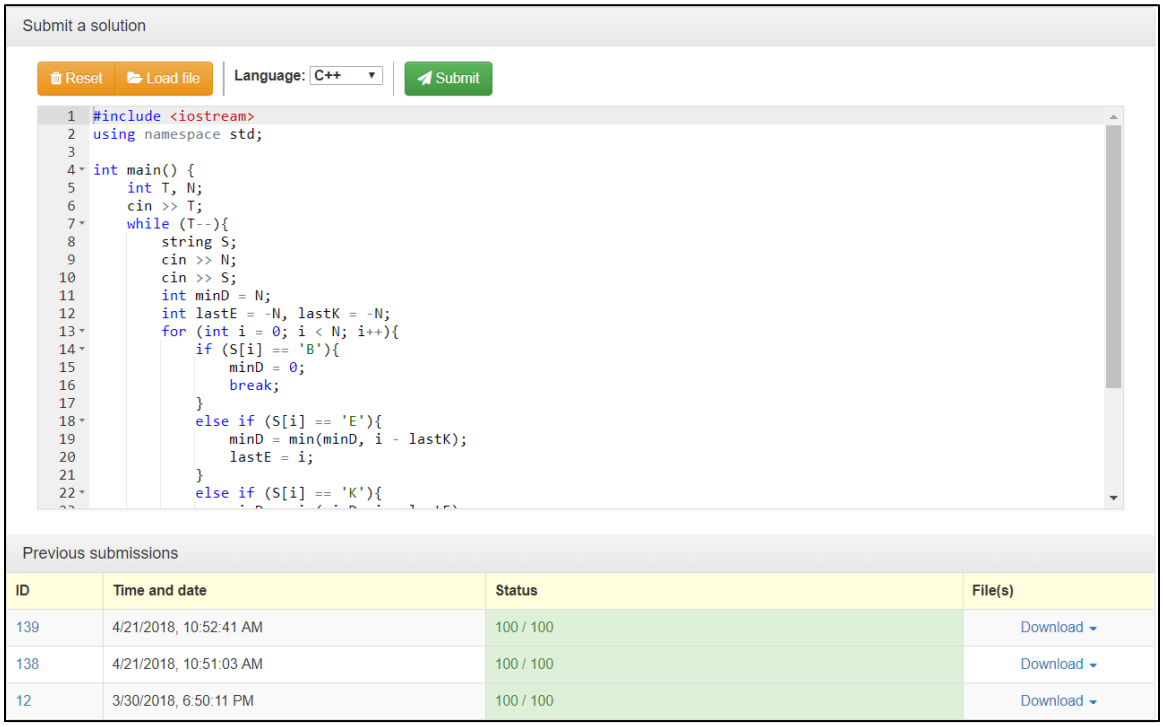
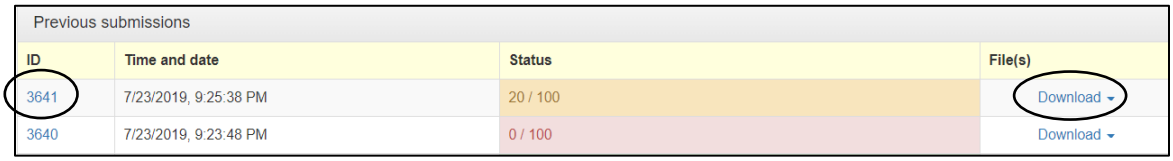


Figure 24: Code editor and previous submissions

The students can submit their solutions and view their current/previous submissions and scores. When submitting a solution, the students are presented with their total score for the task. Students can review detailed feedback on how their solution handled each test input data associated with the selected task if they require additional information. Each submission is stored on the server with a timestamp containing submission data. Students and teachers can download solutions through the administrator panel (Figure 25).


Previous submissions

ID	Time and date	Status	File(s)
3641	7/23/2019, 9:25:38 PM	20 / 100	Download
3640	7/23/2019, 9:23:48 PM	0 / 100	Download

Figure 25: Previous submissions for a selected task

The students can view their results in detail if they wish. The feedback for each test case is presented along with the awarded score and the corresponding execution time and memory usage (Figure 26). Teachers have the option to change the level of feedback the platform provides, especially when the task involves subtasks. For each subtask, teachers can choose to show students only the correct test cases and keep the incorrect information hidden. Hidden data can prevent students from guessing input sizes of test cases which would be possible if they could view all the related information.


Submission details

Testcase	Result	Details	Time	Memory
000	Not correct	Output isn't correct	0.000s	256 KiB
001	Not correct	Output isn't correct	0.000s	256 KiB
002	Not correct	Output isn't correct	0.004s	128 KiB
003	Correct	Output is correct	0.000s	256 KiB
004	Not correct	Output isn't correct	0.004s	128 KiB
005	Not correct	Output isn't correct	0.004s	128 KiB
006	Correct	Output is correct	0.000s	128 KiB
007	Not correct	Output isn't correct	0.004s	128 KiB
008	Not correct	Output isn't correct	0.008s	256 KiB
009	Not correct	Output isn't correct	0.008s	256 KiB

Compilation output

Compilation outcome:	ok
Compilation time:	0.264s
Used memory:	37.6 MiB

Figure 26: Full feedback for a task

Tags: The tags associated with the selected tasks. With the corresponding tags, each task is categorised under specific topics. Several tasks can be found under two tags.
Time limit: The time limit for the selected task for the execution of a single test case measured in seconds.
Memory limit: The memory limit for the selected task for the execution of a single test case measured in megabytes (Figure 27).

Figure 27: Task tags, time and memory limits

One of the platform's main advantages is the ability to accept multiple submissions from multiple users in a contested environment for an extended period without lengthy response times or delays in submissions. The evaluation and scoring engines have been implemented and tested with a simultaneous massive volume of submissions at the final stage of development. The platform's submissions database was able to hold the stress test quite easily. All of the process management of the platform can be done through the teacher/administrator panel.

4.7.2 Teacher-level interactions

I can perform the platform administration on two levels. On the first level, the other teachers and I can perform various tasks from within the platform itself. In this way, a teacher can use the platform as a regular user and simultaneously add course material. Course material includes lesson notes, additional notes (images, scripts, visuals) and quizzes for a specific course topic (Figure 28).

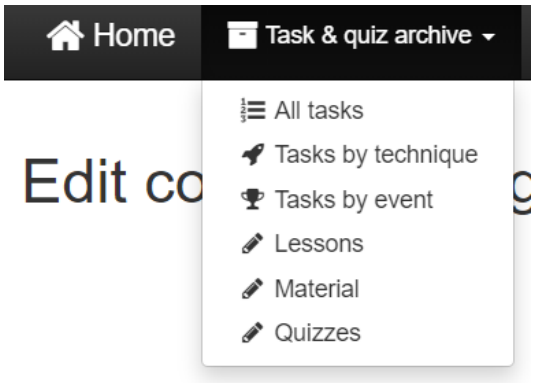


Figure 28: Administrator menu on platform

The platform has enabled the other COI teachers and me to keep the course content decentralised and categorised, which was never achieved within the five districts. The course content was distributed and accessed, and it was crucial for students' progression throughout the course curriculum within a single school year.

There is a separate administration panel for creating and handling the programming tasks, which is the second and more advanced level of the platform's administration. The administrator's panel is identical to the CMS panel, and it permits specific users (teachers/administrators) to perform the administrative processes and tasks. These tasks include monitoring the platform's status and managing the platform's contests, tasks, users and teams.

Monitoring includes three separate categories: queues, workers and logs. During a competition, specifically through the last stages, the students' submissions are continuous and frequent. An administrator must keep track of the submissions queue and ensure that the system evaluates all submissions even after the contest is over. Monitoring workers is a crucial task. The workers are responsible for running the compilation and evaluation of the submissions in a robust setting. I used a setup of four workers for the Michanicos platform and fifteen workers for the CMS server. This particular setup has enabled all of the local competitions to run smoothly. For supporting the monitoring, I supervised system logs to enable administrators to keep track of the current status of the workers and the system services and perform necessary actions whenever the circumstances required.

The administration panel allows total control over contests, tasks and user management. Multiple contests can be run and administered simultaneously. The administrators can control contest specifications such as the contest public URL, contest tasks, programming languages allowed, eligible users, contest duration, scoring format and the maximum number of submissions per user. They can create and add tasks for each contest through the panel effortlessly and efficiently. The administrators have to prepare and upload the task statement in pdf format for each task. Then the task type is selected, which defines the scoring format.

In competitive programming, the task type can be one of the following distinct types:

1. Batch tasks:

Batch tasks are the most common task types in programming competitions. The programme reads its input from the keyboard (standard input) and outputs the results on the screen (standard output). For a batch task, all of the input data is accessible from the beginning of the programme execution, and it is independent of the programme's performance. For this specific type in IOI competitions, a grader function is used, which receives all of the input data in function arguments and returns all of the output data.

2. Interactive tasks:

In an interactive task, some data must be generated before new data becomes accessible. Accordingly, the data input may be dependable on the previous data output. The programme has an exchange with the grading system, which may behave like a challenger. The task is interactive when the input is predefined but not entirely accessible at the beginning of the execution.

3. Output-only tasks:

In output-only tasks, the participants are not required to submit source code but only the output files corresponding to specific input files. Indeed, the construction of these output files involves the use of strategies and, in almost all cases, requires substantial volumes of code. The evaluation of the submissions for this type of task does not include programming languages or programme compilation and execution. Nevertheless, there is no evidence of the algorithms or strategies the contestants have created.

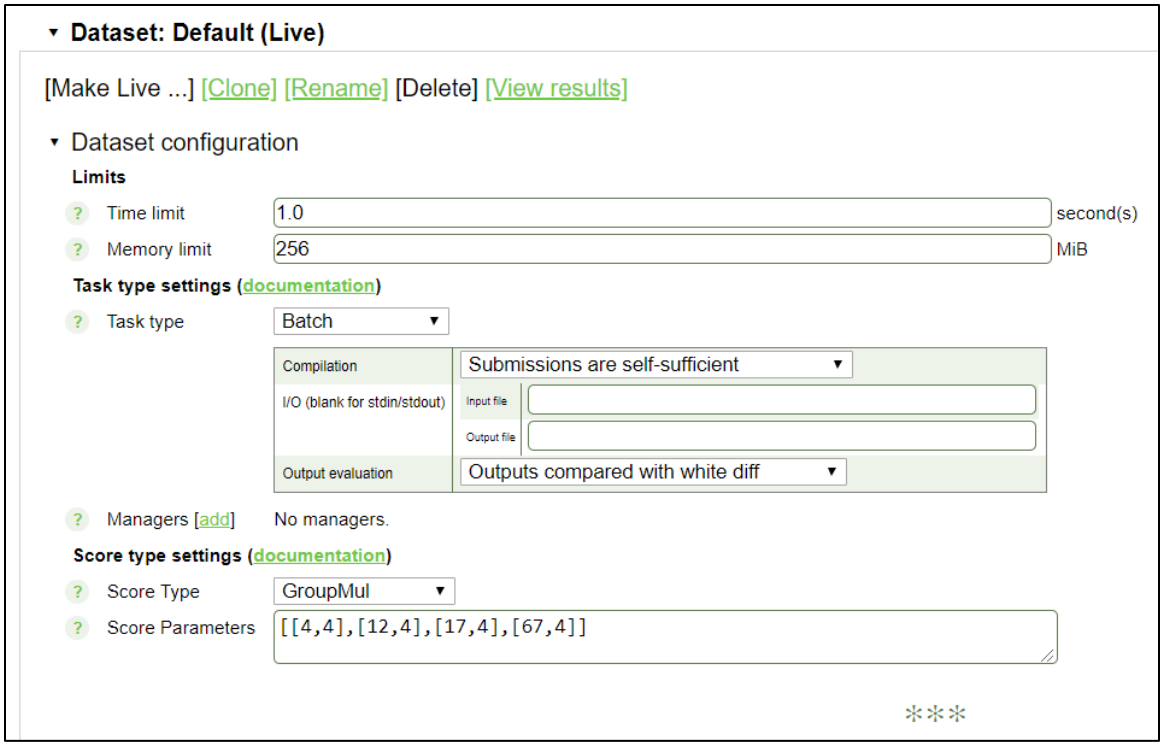


Figure 29: Task setting

After selecting the task type, the administrator must set the time and memory limits and select the scoring format. In the simplest type of scoring, a value parameter is set for each test case, and the cumulative score is the addition of the scores of the correct test cases. In tasks that include multiple subtasks, the scoring format is arranged in groups (Figure 29). For example, if there are five test cases in the first subtask and the points allocated with the subtask are fifteen, the student must solve all five test cases correctly to be awarded the fifteen points. For solving correctly four or fewer test cases, no points are awarded.

The final step in the task setting process is the addition of the test cases. The test cases need to be checked for consistency before adding them to the platform, but additional checks can be performed after uploading the data. Multiple solutions can be submitted for testing purposes to verify the validity of the test cases, and any inconsistent data must be removed and replaced. The ideal number of test cases for a programming task is between 10 to 50. However, several IOI tasks require up to one hundred test cases. The platform's panel supports uploading multiple test cases in the form of a zip file which is convenient and efficient when there are numerous tasks to be uploaded simultaneously (Figure 30).

▾ Test cases

Add a testcase Add multiple testcases Download all testcases

#	Codename	Public	Input	Output	Ops
1	test11	☑	Show input	Show output	Delete
2	test12	☑	Show input	Show output	Delete
3	test13	☑	Show input	Show output	Delete
4	test14	☑	Show input	Show output	Delete
5	test21	☑	Show input	Show output	Delete
6	test22	☑	Show input	Show output	Delete
7	test23	☑	Show input	Show output	Delete
8	test24	☑	Show input	Show output	Delete
9	test31	☑	Show input	Show output	Delete
10	test32	☑	Show input	Show output	Delete
11	test33	☑	Show input	Show output	Delete
12	test34	☑	Show input	Show output	Delete
13	test41	☑	Show input	Show output	Delete
14	test42	☑	Show input	Show output	Delete
15	test43	☑	Show input	Show output	Delete
16	test44	☑	Show input	Show output	Delete

Figure 30: Adding test cases to a task

The administrator has complete control over the users' profiles and submissions. There are multiple advantages to keeping all of the submissions on the server. For checking each student's progress and keeping track of their learning trajectories on the platform, special attention was given to the correct setup of the backup system on a separate drive. The teachers have the option of retrieving all the submissions for a contest, for one specific task, or all the submissions of individual users (Figure 31).

Time	User	Task	Status	Files	Token	Official	Comment	Reevaluate
2019-08-12 12:32:31.069587	adamos2468	Infinity	Scored (20.0 / 100.0)	Infinity.cpp	No	Yes		C E S
2019-08-12 12:14:19.931958	adamos2468	Infinity	Scored (0.0 / 100.0)	Infinity.cpp	No	Yes		C E S
2019-08-12 12:03:34.921387	adamos2468	Infinity	Scored (0.0 / 100.0)	Infinity.cpp	No	Yes		C E S
2019-08-12 11:57:35.409610	adamos2468	Infinity	Scored (0.0 / 100.0)	Infinity.cpp	No	Yes		C E S
2019-08-11 11:38:22.139190	michalis001	Infinity	Scored (0.0 / 100.0)	Infinity.cpp	No	Yes		C E S
2019-08-06 15:03:28.476188	Dremix10	Trucks	Scored (40.0 / 100.0)	Trucks.cpp	No	Yes		C E S
2019-08-06 11:09:58.584546	Dremix10	Trucks	Scored (20.0 / 100.0)	Trucks.cpp	No	Yes		C E S
2019-08-03 10:36:07.359887	marcoskal	Trucks	Scored (0.0 / 100.0)	Trucks.cpp	No	Yes		C E S
2019-08-03 10:20:17.359750	marcoskal	Trucks	Scored (10.0 / 100.0)	Trucks.cpp	No	Yes		C E S
2019-08-03 07:29:16.824336	marcoskal	Trucks	Scored (40.0 / 100.0)	Trucks.cpp	No	Yes		C E S
2019-08-02 13:23:39.580792	Panas	Trucks	Scored (50.0 / 100.0)	Trucks.cpp	No	Yes		C E S
2019-08-02 13:23:14.430672	Panas	Trucks	Scored (10.0 / 100.0)	Trucks.cpp	No	Yes		C E S
2019-08-02 12:56:10.990765	Panas	Trucks	Scored (0.0 / 100.0)	Trucks.cpp	No	Yes		C E S
2019-08-02 12:51:24.245914	Panas	Trucks	Scored (60.0 / 100.0)	Trucks.cpp	No	Yes		C E S
2019-08-02 12:35:20.334740	Panas	Trucks	Scored (40.0 / 100.0)	Trucks.cpp	No	Yes		C E S
2019-08-01 05:28:20.790995	Theo830	lines	Scored (0.0 / 100.0)	lines.cpp	No	Yes		C E S
2019-08-01 05:27:58.379325	Theo830	lines	Compilation failed	lines.cpp	No	Yes		C E S

Figure 31: Viewing all submissions on the platform

The source code for each submitted solution can be viewed on the screen or downloaded on a local machine for testing purposes. For each user's submission, there is a timestamp

indicating the specific date and time of the submission. Thus, the teachers can estimate the time frame between the initial and the final submission, scoring 100 points. In case of an error in the test cases, the administrator can re-evaluate and re-assess all the submissions for one specific task.

For each separate contest during the specified contest time frame, users can submit questions through the platform requesting clarifications for the tasks (Figure 32). Through the panel, the administrator can reply privately to each user using a set of predefined answers: *Yes / No / No comment / Answer is in task description / Invalid question*. Moreover, the panel offers an additional announcement feature that provides real-time public messages to all users during contest time.

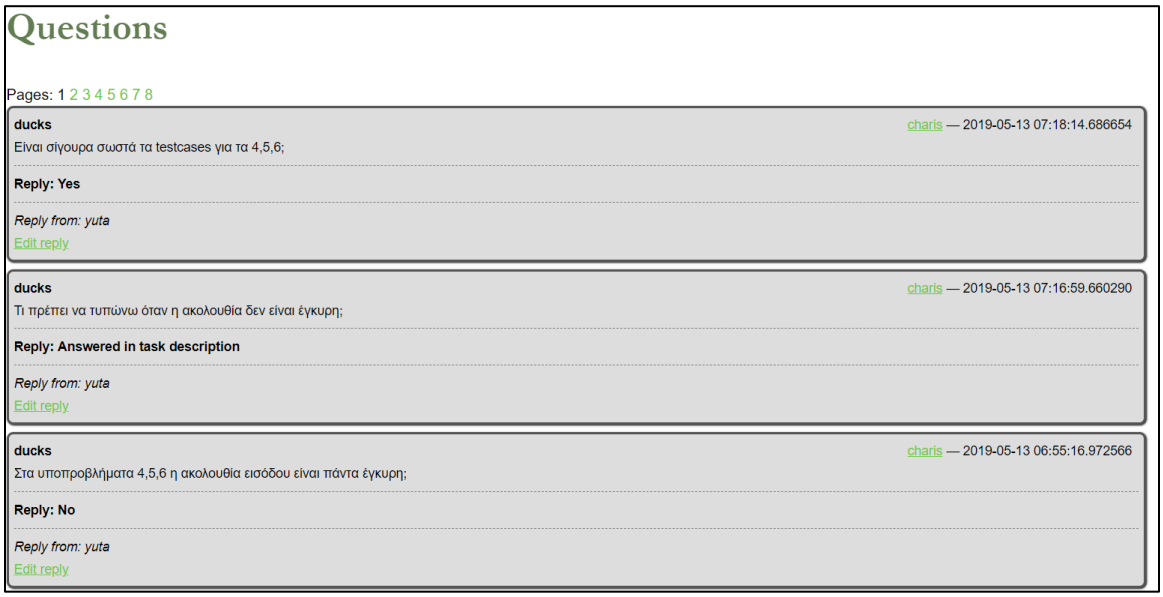


Figure 32: Q&A feature during a contest

4.8 Research study activity: Slack interactions

I promoted and encouraged communication among students and teachers using the Slack workspace. Students were encouraged to discuss their progress or stuck points with their teachers and other students. The Slack platform enables users to communicate privately or through groups (channels). Slack was used in parallel with the platform while the platform's forum component was under construction. Slack offered the COI teachers a variety of tools to use when sharing information with students or peers. It has been an

inseparable component that has successfully complemented the Michanicos platform interactions most effectively (Figure 33).

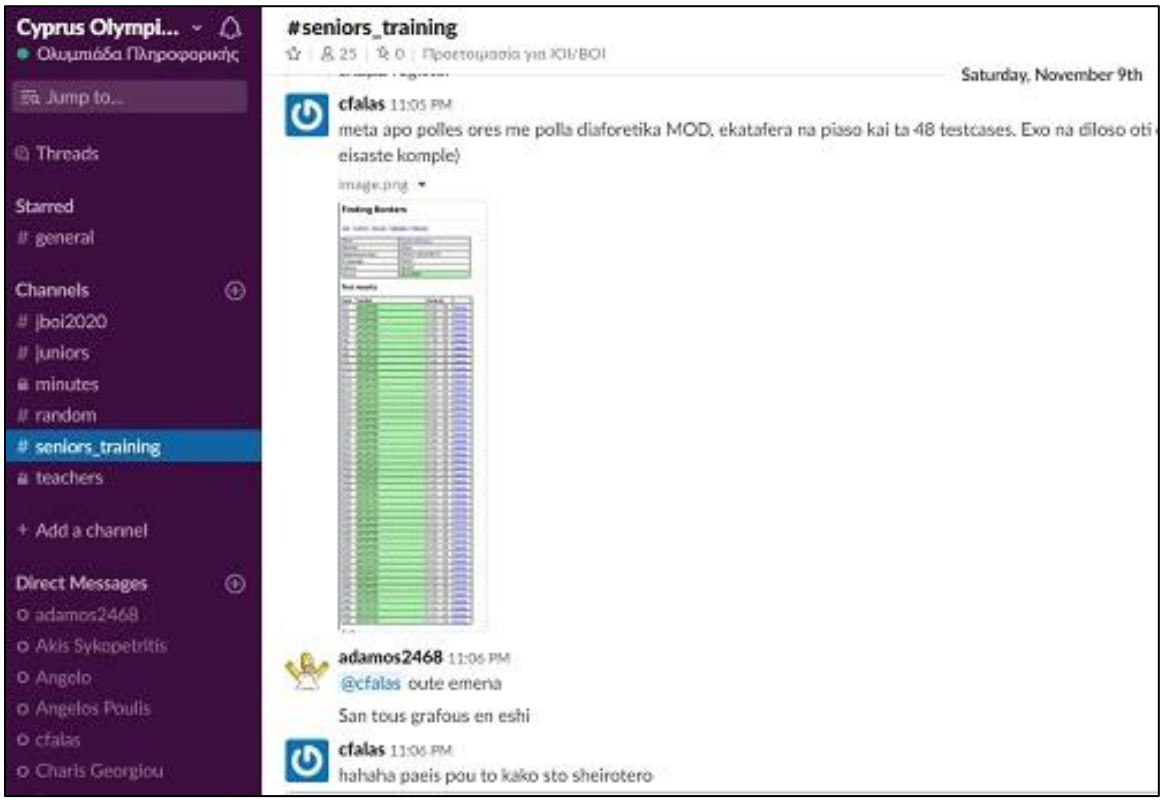


Figure 33: Slack workplace interactions

Slack enabled teachers to organise students into channels (groups of students) according to their programming level and share information and material accordingly. I created five separate channels during the study: jboi2020, juniors, seniors, minutes and teachers. I created the 'seniors' channel specifically for the IOI/BOI delegations and third-round contestants. In this channel, previous IOI tasks and the most advanced programming tasks on Michanicos were discussed and analysed. The feedback from the teachers/instructors was as minimal as possible. In most cases, only one hint per task was given. I created the 'juniors' and the 'jboi2020' channels for the EJOI/JBOI delegations and second-round contestants. In these channels, the tasks that appeared in previous EJOI and JBOI competitions were discussed among students and teachers. The feedback from teachers was more elaborated than the advanced group. Usually, two or three hints per task were given. In some situations, students who successfully solved a task were encouraged to offer hints to other students.

The ‘minutes’ and ‘teachers’ channels were created for teachers and alumni to keep the action learning set connected throughout the study. In these channels, I presented the agendas from the analysis sessions and discussions with colleagues aimed for interventions to align the curriculum and tasks’ assignments among all districts. Finally, the general channel was the default channel where all users had unlimited access. It was used for discussing the introductory material covered during the first and second rounds. In this channel, the feedback was maximal. In some cases, the full spoiler was given to students. At different times, collaborations between two students struggling with the same task were also encouraged. Most of the additional material was shared in this channel so that everybody could access it. The additional material shared among students and teachers included visualisations, images and files.

To ensure that all communication was monitored and to safeguard the proper interactions among students, I set the following rules:

1. No spamming of the channels was permitted. To maintain communication similar to the competitions’, the student requests had to be as accurate as possible. Requests such as ‘How do I solve this task?’ were requested to be rephrased in detail: ‘Is the Dijkstra’s algorithm (Dijkstra, 1968) appropriate to solve this shortest path task?’.

2. No code sharing was allowed between students. This was secured through the platform’s settings. Students could discuss their approach to solving a task but not share their source code. On some occasions, only in the general channel, students were requested to privately share their code with their instructors to offer guidance or help them locate the errors.

3. Students were encouraged to install the Slack application on their smartphones for the entire course. All communication between teachers and students was accomplished securely through the Cyprus Olympiad in Informatics workplace, available at coinformatics.slack.com.

Slack was one of the few applications without restriction rules in any country recently organising the IOI. However, there were several issues with Facebook, Messenger, Twitter, and Google platforms. Using these applications was strictly prohibited in some host countries, and communication with students and teachers was limited.

4. Usernames (handles) were requested to be identical between the Michanicos platform and the Slack workspace so that all code submissions could be easily associated with the help requests.

Collaboration among students supported maximising their own and others' learning and established COI as a truly collaborative learning environment. Social negotiation enabled students to share their experiences in liminality and deal with tension.

4.9 Chapter summary

The purpose of this chapter was to present the main activities of the project study. The emphasis was on the interactions of teachers and students with the framework's components. The Michanicos platform has been integrated into the learning environment and was established as a competitive learning environment for students and a powerful tool for creating, assigning and assessing tasks for teachers within the COI community. The most significant capabilities of the platform are presented and analysed in this chapter.

CHAPTER 5: Project Findings

5.1 Introduction

With this research project, I have made three distinctive contributions to knowledge. I identified threshold concepts for competitive programming, presented a methodology for their identification, and introduced the COI framework. The COI framework is a system of training that encompasses four critical components for improving the teaching and learning processes and providing structured context management in a competitive learning environment. I have acquired substantial evidence for the authenticity of the framework by inquiring about the positive and negative aspects of a collaborative environment. I have used action research to successfully integrate the COI framework into the programming course and evaluate its impact on learning. Due to their nature, the four research objectives were better addressed using a combination of action research and mixed methods approach. As a result, I have adopted a pragmatic perspective on scientific research (Creswell and Clark, 2011; Morgan, 2014). Qualitative and quantitative methods were crucial for proving tentative claims of knowledge acquisition. I consider the selected research methodology highly appropriate, and I am confident that the study's results are valid within the context they were produced and how they were thematically interpreted.

Thematic analysis is a form of qualitative analysis that provides a detailed representation of data not as a specific technique but as a means to use across different techniques (Boyatzis, 1998). By utilising thematic analysis in action research, a researcher can associate the notions and ideas of students and compare these with the empirical data that has been collected in various situations at various times during the research project (Ibrahim, 2012). In many research projects, there is a tendency to utilise available software packages such as NVivo or SPSS for collecting, organising and analysing the data. However, statistical software packages may not be beneficial, particularly when I need to assess students' programming knowledge in terms of qualitative improvements in the creative manner in which these emerge within their programmes' source code. In such cases, it is more appropriate to use manual analysis rather than software-based methods

and evaluate computer programmes independently. Qualitative research must provide explanations and be consistent with the evidence gathered. Given this, thematic analysis can help identify variables that affect any notion raised by the students. Therefore, the students' interpretations are significant in providing appropriate explanations for their ideas and actions. The last statement follows the ideas generated by the thematic analysis process, which focuses on the interpretation of data and the theory that emerges (Bryman, 2001).

Action research enables the researchers to join the actions taken within the research and relate them to the methodology while using reflection to deliver a solid foundational base for the research study (Coghlan and Brannick, 2005). Through action research, I have evaluated the framework and its effects on learning from multiple perspectives. First, I used empirical research tightly connected to the research objectives with a pragmatic mixed-methods approach to produce accurate data regarding the framework's validity. Then, I produced quantitative and qualitative data to assess the students' code-writing ability and strategies, as well as their level of engagement. This chapter is an empirical evaluation of the proposed framework and its integration into the learning environment of the COI community specifically designed for this research study.

When considering new pedagogical approaches or new educational tools, many parameters should be considered, such as educational context, teachers' views and stakeholders' beliefs. Therefore, it is improbable to integrate all components into one specific training method. Doing empirical research for improving the learning outcomes of education is complex and demanding. I have studied multiple reports in the literature about proposed pedagogical approaches that involved 'students doing something and loving it, and that is why everybody should try it'. Significantly, most education research is evaluated in the context of our classrooms and our teaching. As expected, most of the empirical data in these reports are predominantly positive, and the proposed approach is considered to be effective. As a teacher/researcher, I must thoroughly analyse and investigate how and in what way a proposed pedagogical approach or educational tool works and precisely in what context before implementing it into my practice and expecting miraculous results overnight.

The approach in which the action researcher reports on the findings can establish a level of credibility for the research project. Losing credibility as a researcher might harm future research projects (Coghlan and Brannick, 2005). The findings illustrate how the action research project is relevant within the learning community and how it may relate to others confronting the same issues. To present the findings, I have considered two approaches reported by Denscombe (2010) to produce more efficient outcomes for qualitative data analysis. One of the approaches suggests concisely compacting extensive and varied data using graphs and tables. This approach enables the proper identification of the findings upon which to emphasise. The other approach suggests making the connection between the research objectives and the findings clear and concise so that the selected research methodology is justified (Denscombe, 2010).

I have collected empirical data using action research cycles to improve my learning community. Action research findings provided new ideas that directed action to enhance the framework's effectiveness and to impact the practice of the action learning set participants. The findings are separated into four specific categories: the pre-liminal variation, the threshold concepts identification, the level of engagement and the programming strategies involved. I have considered the following:

- Students' pre-liminal variation
- Threshold concepts in competitive programming
- Students' perceived and actual coding efficiency with the course material and the identified threshold concepts
- Students' qualitative code optimisations and programming strategies
- Students' feedback on the framework and its components
- Students' level of engagement with the code-evaluation platform
- Statistical analysis of all competition rounds
- Performance assessment of delegations in international programming competitions
- Students' post-liminal variation

The above findings are presented in the following sections, and they are thoroughly interpreted and discussed in chapter 6.

5.2 The pre-liminal space

As previously discussed, investigating the pre-liminal space is critical to determine how students successfully negotiate the liminal space of understanding (Meyer and Land, 2006). Moreover, student engagement is a very significant component of the framework as it formulates the students' learning trajectory. The first questionnaire (Appendix 1) was administered to the Bebras competition top-ranked students to investigate the pre-liminal space in terms of computer literacy, prior programming knowledge, competition performance and willingness to participate in COI lectures and future programming competitions. Students were required to specify their level of agreement on a five-point Likert scale for each concept. A total of 78 questionnaires were completed (52 boys, 26 girls, ages 11-15). The empirical data had to provide insights not only on previous programming knowledge but on the students' epistemological stance and attitudes towards learning. The Bebras competition is a prerequisite of Olympiad participation and is the first step in the recruiting process. It involved a set of multiple-choice algorithmic questions of increasing difficulty designed to challenge the students' ability for solving problems. Evaluating the students' performance was correlated with their disposition to face even more challenging programming tasks in the future. Such tasks may seem unsolvable at first but solvable after a substantial amount of effort and time. Accordingly, the inclination to be physically present at the lectures and the asserted level of engagement within the COI community of learning completed the formulation of an indicative set of pre-liminal space data.

1. High computer literacy among students

Computer literacy refers to the level of expertise and understanding of an individual to operate a computer. The term is concerned with the use of applications rather than the individuals' ability to programme. 92% of the students who participated in the survey reported high computer literacy and the ability to use a computer without the help of an adult.

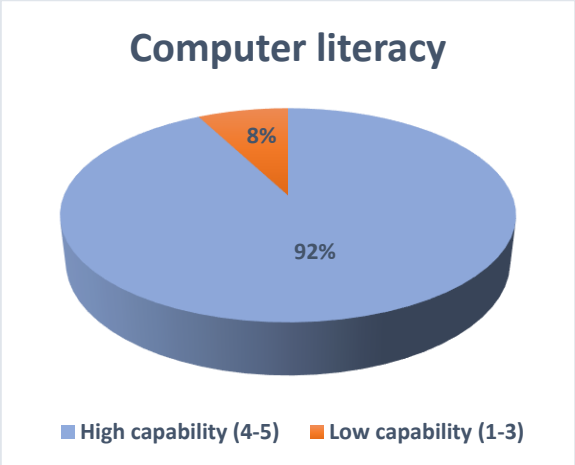


Figure 34: Computer Literacy

Computer usage is a necessity in their daily activities for completing their homework as well as for leisure and entertainment. The European Commission's statement has made it evident that computer literacy has become a vital life proficiency. The incapacity to use or access technology will eventually become a struggle for social integration and individual development (EC, 2008). Cyprus has recognised the importance of computer literacy through our schools, and government funds were used to ensure that every student in secondary education owned a personal laptop computer by the age of 14. The laptop acquisition was significant in ensuring high computer literacy among secondary education students (Figure 34).

Computer literacy can play an important role in programming competitions. During a programming contest, computer literacy regarding the software or the hardware can save valuable contest time. In the unlikely event of a software or hardware failure, the technical committee will assist the students in need. However, when there are frequent help requests from a large number of contestants, the ones that are capable of identifying and fixing the issue without having to wait for the technical committee have a definite time advantage over contestants that rely only on their programming skills. Though computer literacy was high among the COI newcomers, the framework provided additional information about the contest environment and basic software/hardware troubleshooting before each competition round.

2. Prior programming knowledge

Even though prior programming knowledge is not required to attend the lectures at COI, most of the newcomers (59%) have reported knowing at least one programming language. The students who reported having no prior knowledge of programming are primarily students from elementary schools in which no programming curriculum is present. The gymnasium curriculum introduces Scratch in the first grade, a programming language designed to help young students learn

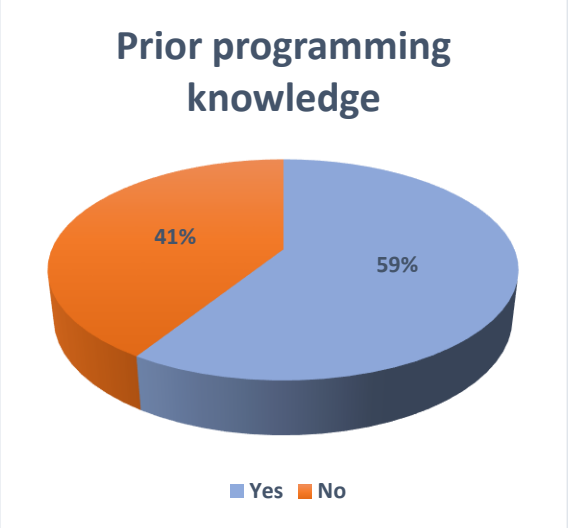


Figure 35: Prior Programming Knowledge

algorithmic thinking. Moreover, the software Alice was introduced in the second grade. Alice is also a block-based programming environment that enables students to design interactive worlds and learn through creative exploration. There is also a mandatory programming Pascal course in the third grade.

As a result, a percentage of 59% of students (Figure 35) reported having prior programming knowledge, with the Pascal programming language ranked first. However, 23% of students reported knowledge of Python (Figure 36) even though this programming language is not taught in gymnasium curricula. Most of these students are self-taught, which is a significant advantage of Python. It is considered the most popular introductory programming language because of its simplicity and the amount of available online material. Young students who want to learn basic programming by themselves can do so very easily by finding one of the hundreds of available Python tutorials online. The popularity of Python has drawn the attention of the IOI International Committee, and there have been initial discussions for introducing Python as one of the official languages of future IOI competitions.

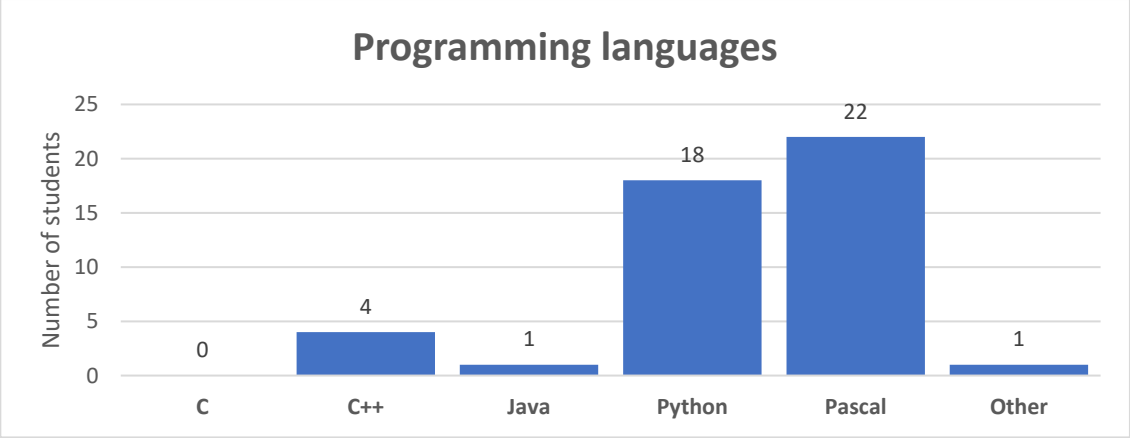


Figure 36: Programming languages identified

Prior programming knowledge can help newcomers adjust to the demanding COI programming course faster and smoother. The transition from Pascal to C++ is effortless, verified by the same language transition in our lyceums. However, the additional support students without any prior programming knowledge received during the first month of COI lectures was equally important as well. These students, most of them still in elementary school, started their programming journeys with C++, and they were initially

taught the most basic input-output programmes. By the fourth week of vigorous introductory lectures, the newcomers with no prior programming knowledge attended the same lectures as newcomers who reported having prior programming knowledge. The additional support for the students continued for the entire year.

3. Bebras competition performance

The Bebras competition is an international competition organised in more than 40 countries, with more than two million participants (Bebras, 2019). The top-ranked students of the Bebras competition in Cyprus are participating in a COI camp each year where they are introduced to programming and are invited to the COI lectures. 86% of these students reported that they had minimum difficulty when they solved the algorithmic tasks of the Bebras competition. Only 3% of students reported having an increased level of difficulty (Figure 37). The goal of the Bebras competition is to introduce students to the algorithmic type of problems and promote the required levels of abstraction for successfully solving these tasks. Carefully designed tasks are used in two competition rounds, and student performance is evaluated through an online testing environment implemented by the CCS.

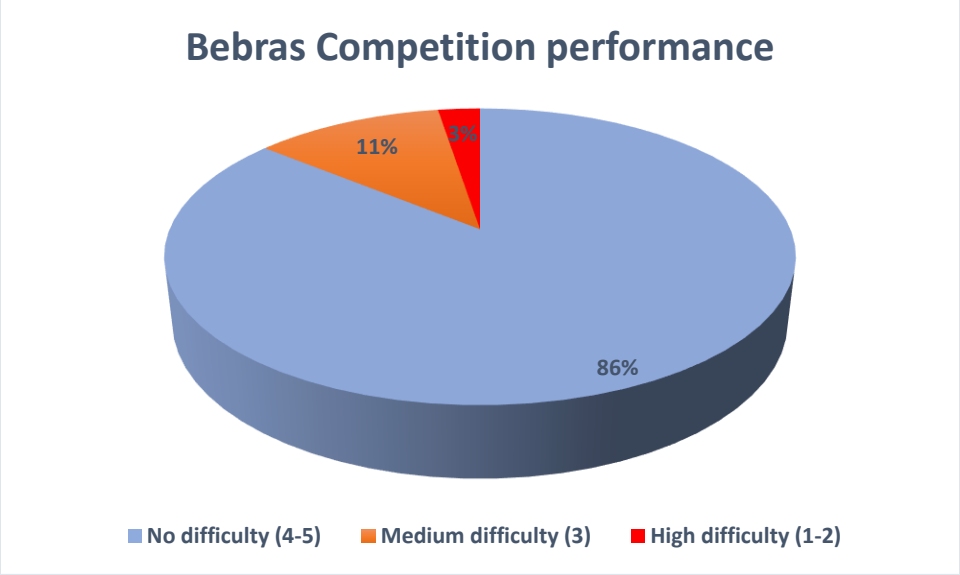


Figure 37: Performance in Bebras competition

To illustrate the level of abstraction required for solving some of the tasks in the Bebras competition, refer to the task 'Spies' (Appendix 12), which was included in the sample tasks. Since most elementary and high school students are used to standardised multiple-

choice tests, this task can be perceived as a threshold concept from their perspective. Clearly, from the images used in the task description, the students were introduced to graphs without even knowing the term. Each spy was a vertex, and each interaction between two vertices was an edge. Some students used paper and pencil to solve it, and several lines were drawn while others used their mathematical background. However, most students found the correct answer. The opportunity from using such problems was to engage students and allow them to attempt to solve these tasks without necessarily grasping their theoretical background. The focus of the competition was to examine the interactions of students with the tasks and determine their reactions when they encountered similar or even more complicated tasks of related nature. The majority of students reported that they were intrigued by these problems. Moreover, they had very few difficulties during the first round of the Bebras competition, and they were looking forward to facing even more demanding challenges.

4. Participation in COI lectures

Participation in the weekly programming lectures is not mandatory to be eligible to compete in the competition rounds. Students have successfully participated in the qualification rounds in the past without attending a single COI lecture. However, the students who qualified for the national delegations have been regular participants at the lectures in all districts. Through lecture participation, students become actively involved with the COI community of learning, and older students become mentors of younger students. I measured students' inclination to participate in the lectures to define their level of determination to engage and give back to the COI community.

Students' willingness to participate in the lectures is an initial indication of the students' commitment to be part of the COI community and become actively engaged with the COI framework. By voluntarily declaring their disposition to join the lectures, 93% of the recruits have taken the initial step of their transformation journey (Figure 38). The first step is often the most difficult one to take, and verifying the informal commitment of the recruits from the five districts was essential for the pre-liminal space investigation.

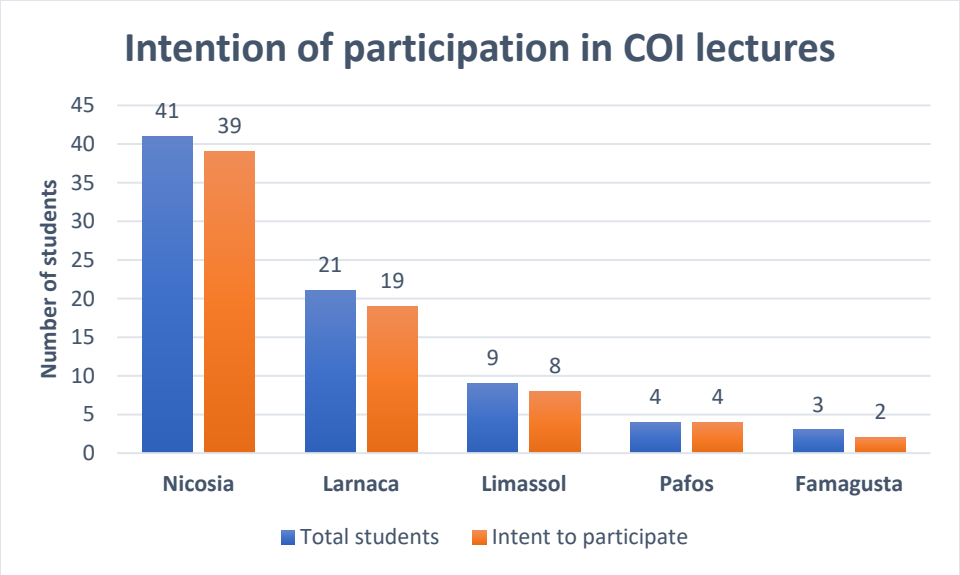


Figure 38: Intention for lecture participation

5. Participation in programming competitions

Similar to the lectures, participation in competition rounds of the COI is not mandatory for students. Newcomers in the past attended every lecture, but they were reluctant to participate in the relatively easy first round of the competition. These students expressed the desire to learn how to programme but did not feel confident participating in competitions. The results of each round are publicly announced, some form of pressure is applied to students to perform satisfactorily. The announcement of four preliminary tasks addressed this issue. The preliminary tasks were announced before the first round, and no score was added to the first-round results. Since one of the preliminary tasks was included in the first-round tasks and the qualifying score was 50%, most hesitant newcomers gained confidence by solving the preliminary tasks and participated in the competition regardless of the outcome.

With the study, I measured the willingness of the top-ranked students of the Bebras competition to participate in future COI programming competitions, eventually representing Cyprus in international programming competitions. As mentioned earlier, the conclusion of the competition rounds is the formation of the national delegations. This is the epitome of our competitive spirit as a learning community, and I try to instil this into my students before their first competition participation. The results for each of the five districts were positive for the Bebras contestants, as 78% responded that they were willing

to participate in future COI competitions and represent Cyprus in international events (Figure 39).

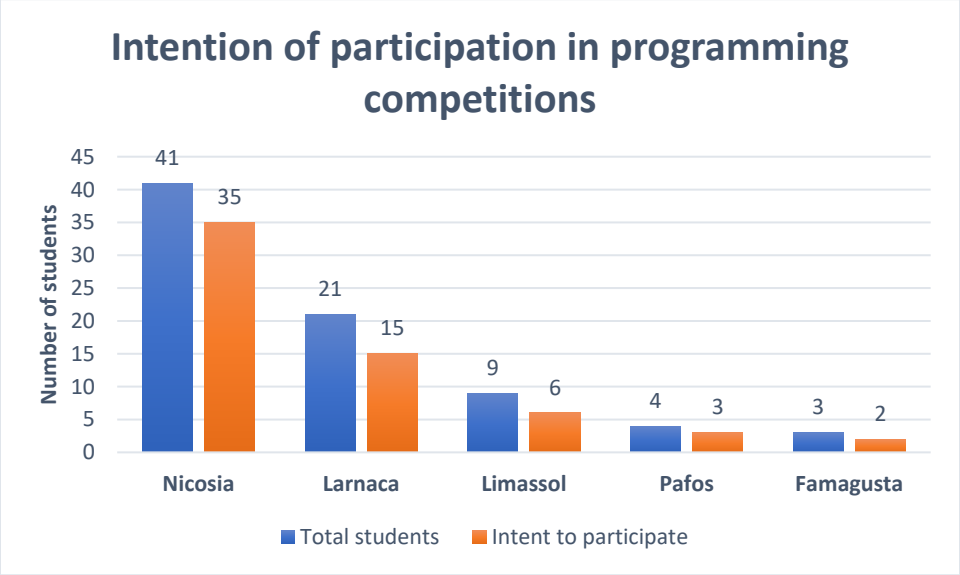


Figure 39: Intention of competition participation

The empirical data provided a promising indication regarding the learning attitudes and epistemological stance of the recruits. The data delivered critical insights on why some students negotiated liminality effectively while others faced more difficulties (Meyer and Land, 2006). The results of the pre-liminal variation yield a significant observation that was key for the following actions. Most of the students that succeeded in the Bebras competition wanted to improve and keep learning. The pre-liminal investigation provided solid indications on what might support or delay the negotiations with liminality.

For investigating the pre-liminal space, I must look beyond the pre-requisites in terms of academic background and focus on the epistemological stance of my students (Rountree and Rountree, 2009). Are they determined to learn the way I expect them to? Are they willing to go through a probably long period of uncertainty and even alternation between appearing to fully understand a concept and feeling as if it will always remain unclear? The students' epistemological stance plays an essential role in self-motivation for self-improvement, which is a crucial element in this framework. Therefore, the pre-liminal variation or acknowledging a student's life stance with an arbitrary level of prior knowledge can help me identify patterns in the liminal variation of students.

5.3 Threshold concepts in competitive programming

As reported in chapter two, there is minimal data in the literature about threshold concepts in competitive programming. Most of the threshold concepts identified within the existing literature are for introductory programming courses, and these concepts are rarely used in competitive programming competitions (Khalife, 2006; Boustedt et al., 2007; Eckerdal et al., 2007; Sanders and McCartney, 2016; Kallia and Sentance, 2017). I used the second method proposed by Davies (2006) to identify threshold concepts within my discipline. Since my goal was to make students think like computer scientists, I had to determine how computer scientists think. I had to study successful practitioners within the field. I chose these practitioners to be past COI competitive programmers who participated successfully in multiple programming competitions and have transformed into aspiring computer scientists. Specifically, my focus was on the distinctive strategies employed by novice students and accomplished alumni for solving the same task.

These comparisons drew upon the first method that Davies suggests for studying the ways practitioners of different disciplines solve similar tasks (Davies, 2006). Admittedly, for an accomplished student and a recruit, competitive programming feels like a completely different discipline. Therefore, I selected the COI alumni as the expert group to identify threshold concepts. Furthermore, after the initial identification of the concepts, my colleagues from the action learning set and the students from the focus group confirmed that the identified threshold concepts of this research were indeed the most challenging to teach/learn.

The second questionnaire (Appendix 2) was administered online, as most alumni were studying abroad to identify and authenticate threshold concepts. They have participated in international programming competitions, and three have won medals. Eleven of them have pursued a degree in CS and the other two in Mathematics. Due to the qualitative nature of the questionnaire, the results were more time-consuming to analyse. A concern was whether I would have attained similar data if I had retrieved their responses while they were still negotiating liminality rather than after they had gained an understanding. To address this issue, a third questionnaire (Appendix 3) was administered to fifteen COI students who have qualified for the third round of the competition. To assess the threshold

concepts identified with formal measurements of achievement, I investigated the students' perceptions about their coding efficiency and their negotiations with liminality and identified threshold concepts along with their feedback and interactions with the Michanicos platform.

The topic of dynamic programming (DP) is initially introduced in the second round of the competition, with most of the competition tasks being straightforward (Coin change, Minimum/Maximum Sum, Longest Increasing Subsequence, Knapsack). The topic gets gradually more complicated, specifically when DP optimisations are introduced in the third round of the competition. The concept of DP was ranked as the hardest to understand from the alumni responses, and, accordingly, it produced the lowest reported perceived coding efficiency (13%) among the current students (Section 5.4). From the initial alumni responses, DP ranked first in the threshold concepts identified. In contrast to other topics, it is a highly sophisticated technique and not an easily taught algorithm.

A notable 77% of the alumni reported DP as the single subject they faced the most learning difficulties (Figure 40). Segment trees ranked second with 15%. DP tasks regularly appear in both competition days in the International Olympiad in Informatics and other international programming competitions. It is generally believed among IOI team leaders that if students are well prepared on this specific subject, they will end up winning a medal in the competition.

What makes DP a complex topic is that there is no simple way to learn it. It is not an algorithm that can be memorised, so no predefined pattern can be followed to solve DP tasks. Therefore, students need to discover new ways of approaching these tasks as prior knowledge is insufficient. In all my years of teaching competitive programming, the first indication of students successfully altering and improving their ways of thinking has always been their understanding of the concept of dynamic programming.

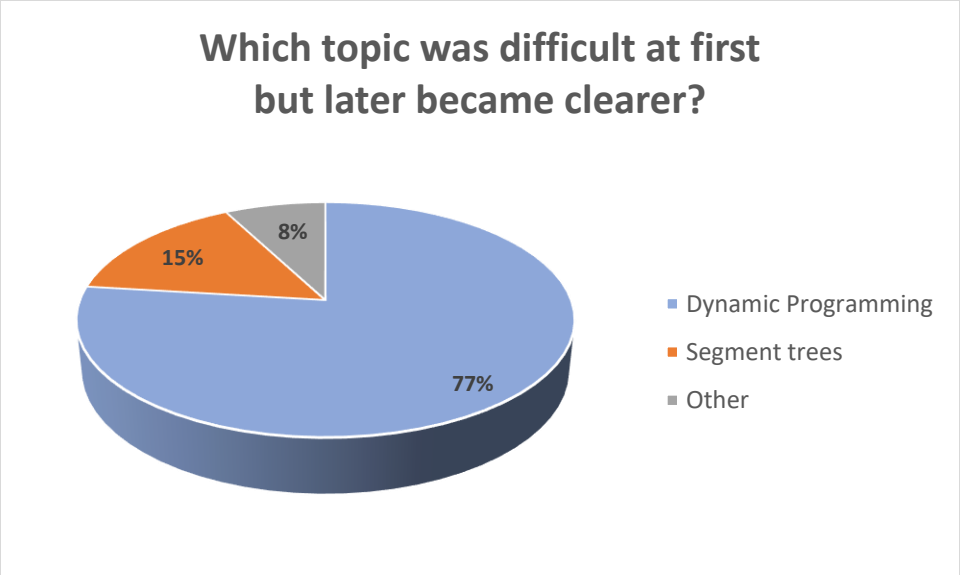


Figure 40: Threshold concepts identified

As mentioned in chapter two, the concept of DP meets all of the threshold concepts characteristics. When students are initially introduced to the notion of DP, they struggle to grasp it, and they are unable to code any suitable DP implementation; therefore, DP is troublesome. Moreover, once the students come to terms with the notion of DP, they demonstrate a significant transformation in their mental process; therefore, DP is transformative. Once the notion of DP is grasped, it cannot be unlearned; therefore, it is irreversible. DP is integrative because it enables students to create new mental connections and notice previously unfamiliar relationships with other programming techniques such as recursion. Lastly, DP is bounded as it is exceedingly invaluable in competitive programming education; nonetheless, it is a very significant theoretical concept that defines what can be calculated. Similar characteristics can be found in the other threshold concepts identified by the alumni.

Meyer and Land (2005) refer to the liminal space when dealing with a specific threshold concept as problematic and humbling for the students. They also suggest that students may experience frustration and anxiety while learning the threshold concepts (Meyer and Land, 2005). I have studied some of the alumni responses from this viewpoint to find evidence of emotionally loaded terms. Several questions on the questionnaire were explicitly aimed at identifying the reactions, perceptions, emotions and attitudes when

dealing with threshold concepts. Several individuals repeatedly mentioned that the learning of threshold concepts was frustrating and humbling.

> *Alumni 2: At first, I hated it and did not want anything to do with it… then I could not do without it.*
> *Alumni 4: It was challenging at first. I felt frustrated and disappointed. I struggled to solve even the most trivial tasks.*
> *Alumni 6: Initially, I solved the easy tasks, but after that, I could not use the same technique for other similar tasks. It was disappointing and discouraging because other students could solve these tasks without real effort.*
> *Alumni 8: …very confusing first encounter but made sense after a while.*
> *Alumni 9: My initial reaction was: I cannot write Chinese. Then I became 'Chinese'.*
> *Alumni 12: Learning this was very disheartening in the beginning. I have previously won medals in Mathematical Olympiads, and I consider myself a good problem-solver, but I could not even write two lines of code for this topic.*
> *Alumni 13: I thought it was the most complicated thing I have ever encountered. Nothing made any sense. I remember, at one point, I thought to myself: I cannot do this. I will never be a good programmer.*

It was interesting to discover how crossing liminality was experienced differently and observe the particular emotions associated with the successful passage. The alumni's perceptions of having negotiated with liminality successfully were as if they were able to visualise or 'dreamt' their solutions to complex tasks before writing a single line of code.

> *Alumni 2: I got stuck quite a lot… But later, as soon as I came across a task that required this technique, I could create an image in my head that illustrated the way to go. It was almost automatic after a while.*
> *Alumni 4: Nothing made sense… My initial disappointment helped me try harder. Then, I could 'see' the solution and write it down on paper before coding.*
> *Alumni 8: Once, I struggled with a hard task for days. One day, after working on it unsuccessfully until past midnight, I decided to go to sleep. When I woke up the*

next day, I wrote the solution within the next five minutes. I must have dreamt about the solution!

Alumni 9: I was able to draw the memo tables almost as soon as I read the task description. It was as if the solution was stored in my memory.

Alumni 12: I knew most mathematical concepts from before, so I could just run the heuristics of the algorithm in my head and figure out the occurring states.

The experience of the successful passage was relatively transformative and rewarding when they finally understood a threshold concept.

Alumni 1: After I began coming to terms with dynamic programming, though I initially did not attempt it at all, I could tell almost immediately where it was needed or not.

Alumni 4: When I won my first medal by solving a complex task that I struggled with for more than three hours, I felt like I could solve anything...

Alumni 7: I have been able to experience the 'joy of code' that my teacher always mentioned. …the excitement I felt after solving a hard task was something incredible.

I have also examined the time frame spent in the liminal space, the intermediate period between starting to grasp a threshold concept and thoroughly understanding it. The alumni responses for the time needed for the threshold concept to become clear ranged from three weeks to an entire year. The short and extended periods have been explained by Meyer and Land (2005) as the students can sometimes negotiate liminality and the associated threshold concepts in a single 'aha' moment. Still, more often, it seems that students require much longer. Correlation with the other questionnaire responses was critical to determine what has worked more efficiently with students reporting less time for grasping the concepts.

Alumni 5: It took me two months and a lot of individual work. Once I got the basic idea, everything became clearer.

Alumni 7: I began feeling confident in solving tasks after about six months of training.

Alumni 8: I have struggled for approximately three weeks… I spent almost six hours a day coding.

Alumni 13: I spent almost an entire year solving the easiest tasks. Then, I solved a medium difficulty task on Codeforces and then another. I was shocked!

The alumni also reported on ways that eventually helped them grasp the threshold concept and successfully cross liminality. The previous statement is one of the reasons the COI alumni were selected to provide information for the threshold concepts identification. The selected alumni have participated in the COI lectures for a combined 57 years. They have competed in international programming competitions, and they have won medals. Therefore, they are considered an expert group for competitive programming. This group has faced its share of struggles with the threshold concepts and has successfully negotiated with them. I was interested in looking into the approaches they used to overcome the learning obstacles associated with the threshold concepts other than the support of their teachers. Since their training occurred before the development of the Michanicos platform, 53% of the alumni reported that what helped them cross liminality was solving complex programming tasks using online judges. Camp participation was second with 27% and studying the COI website's material, among others, was third with 20% (Figure 41).

Alumni 1: Lots and lots of practice. I solved more tasks than I can remember.

Alumni 3: It took a lot of training. Literally, more than 300 solved tasks for a specific subject.

Alumni 4: …through camp participation and discussions with teachers and other students.

Alumni 10: …websites such as Codeforces, Spoj, UVa and HackerRank provide tons of material and practice tasks.

Alumni 12: Solving tasks on Hellenico and USACO was the way to go for me. I have solved all of the corresponding Hellenico tasks within ten months.

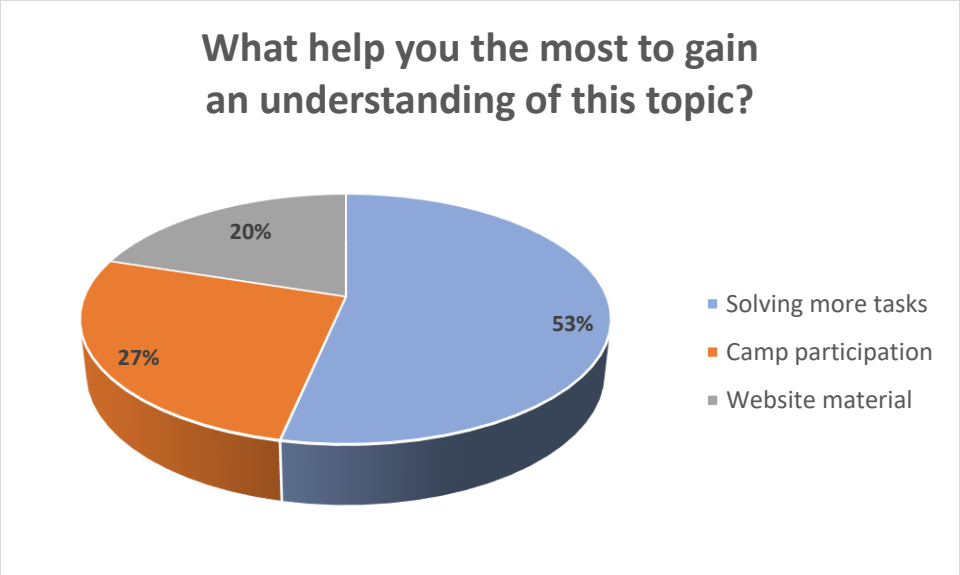


Figure 41: Methods used for negotiating liminality

One of the questions was intended to discover possible other concepts that the alumni needed to understand before negotiating a specific threshold concept. Acknowledging which prerequisites are essential to gain a concrete understanding of an identified threshold concept, particularly those regarding abstraction levels and the measurements on these levels, is pivotal.

What is interesting is that 80% of the alumni that selected dynamic programming as the hardest topic to grasp reported that understanding recursion was critical to understanding the concept of DP. Recursion is a threshold concept identified in introductory CS programming courses (Eckerdal et al., 2007; Rountree and Rountree, 2009; McCauley et al., 2015; Kallia and Sentance, 2017). Therefore, even though only one alumnus selected recursion over dynamic programming as the hardest concept to grasp, it can be safely assumed that recursion is indeed a valid threshold concept in competitive programming as well. Understanding dynamic programming requires a concrete understanding of the rationale and application of recursion, not only why it is essential to know and use it but, additionally, to understand how to apply it when solving new and unknown tasks.

When I asked the alumni to propose training methods they would use with students stuck in the same threshold concept as them, they provided interesting data. Most of them

responded by following an algorithmic step by step approach and lots of examples. What seems to be shared in the responses is that a required level of abstraction must be present before starting to code. The level of abstraction obtained defines the ability to reach an optimal solution quickly and effectively.

> *Alumni 1: …lots of visualisations will do it…*
> *Alumni 3: …same way I learned it. Lots of gradually harder examples...*
> *Alumni 9: …learning the optimal solution line by line and making the necessary mappings when appropriate...*
> *Alumni 10: …make them 'see' the solution in their head before they start typing…*

Primarily they emphasised the rationale of the concept, explaining why you must know this technique and why you should use it.

> *Alumni 5: To learn this technique, you must first understand why you need to use it. Some tasks explicitly require it, and you cannot get an AC without it.*
> *Alumni 8: …for DP, I would initially show them an example with coins. With coin denominations of 1, 3, 4 cents, when you want to get 6 cents with the minimum number of coins, the greedy approach will give 3 (4, 1, 1), which is wrong. The correct answer is 2 (3, 3), which is only obtainable using dynamic programming.*
> *Alumni 11: …you should use it every time you need to maximise or minimise a value… the Knapsack problem is a good starting point…*

Noticeably, the alumni's 'aha' moment was expressed as connecting theory to practice.

> *Alumni 4: …to use the theoretical context of this concept and be able to apply it in practical situations. In my opinion, this connection is the hardest thing to achieve…*
> *Alumni 12: …when you get to the point of recognising why and how to use it, you realise you have reached a different level of knowledge on the subject…*

The latter comment is crucial as it is a valid indication of the students' successful negotiation with liminality. When examining the experiences of crossing liminality, I must acknowledge that they are not as trustworthy as they may seem at first. Some students gain partial understanding and some abstract knowledge, so they believe they have crossed liminality, and they do not seek to gain a better understanding. What if a student is awarded 50 out of 100 points in a task embedded with an identified threshold concept? Can this be considered as partial understanding or not? I had to align the students' perceptions with their teachers' beliefs of what is considered a concrete understanding of a threshold concept. The learning set verified that the successful application of concepts by consistently solving complex unknown tasks could be used as a reliable indication of the successful negotiation with liminality.

Furthermore, these partial understandings probably justify why students get stuck at different points even within the same threshold concept. Thus, it is evident that the path through liminality cannot be considered as a simple linear progression. Additionally, the distinctive partial understandings of not being capable of progressing from an abstract understanding to a solid rationale and application of the concept may be specific to competitive programming and require further investigation. One particular observation from the investigation was the lengthy period of the liminal space negotiations. Even the alumnus who reported the shortest time (3 weeks) stated that he trained for more than six hours daily. Most of the alumni (85%) reported a very long (more than six months) negotiation process with the specified threshold concept.

What was interesting is that the alumni recognised that learning these concepts is time-consuming, and attempting to learn them without dedicating the required time frame can lead to partial and not concrete understanding. The lengthy learning period is a burden for the newcomers as they are not accustomed to dedicating more than a week to understanding their school material. Within the proposed framework, the support for newcomers starts with enforcing the idea that learning takes time, and it is normal for the concepts to be elusive initially. There should be no deadlines, and there should be no immediate pressure on students to deliver results. The learning path is different for each

student, and the support they should receive must be according to the struggles they face.

Another observation was about the vast range of emotions experienced in the liminal space, which is strongly related to the nature of the discipline. The threshold concepts of competitive programming are considered highly complicated. It is common for students to detest or even fear them at first. These initial stressful emotions turn into positive emotions of triumph and accomplishment as students cross liminality. Within a supportive learning environment, no emotion should be dismissed but instead handled and explained (Eckerdal et al., 2007). Even the negative feelings are desirable as they constitute human reactions, and students must navigate through them with the teachers' support. Overall, the initial findings on threshold concepts have helped structure the next steps of the process. There were three threshold concepts identified by the alumni: dynamic programming, segment trees and recursion. The identification of the concepts was followed by communicating the initial findings with the action learning set and discussing proposed approaches for successfully teaching the identified concepts.

The teaching strategy was decided unanimously: use these concepts in several programming tasks within the competition rounds, create new and gradually more complex practice tasks on these threshold concepts and upload them to the Michanicos platform, collect data on students' perceived coding efficiency for all the course material and, finally, assign the tasks embedded with threshold concepts to students and assess/correlate their performance data. The results were then evaluated and discussed in the following action learning set sessions, and future actions were determined.

5.4 Students' perceived/actual coding efficiency

The students who qualified for the third round answered the third questionnaire (Appendix 3). For each of the three rounds of competition, students were asked to provide feedback regarding their perceived coding efficiency for each of the topics found in the course, including the threshold concepts. The other course topics could not be neglected because even if they were not identified as threshold concepts, they might have appeared in an IOI/BOI task. The focus of the action learning set was on the students' responses and

their corresponding interactions with the associated programming tasks on the platform, as well as their performance scores. I coded the responses in the following format, with five levels of perceived coding efficiency:

Your Response	**Level**
Not able to code it at all	1
Not able to code it correctly	2
Able to code it with some help	3
Able to code it with minimal difficulty	4
Can code it efficiently in a competition round	5

Table 6: Perceived coding efficiency levels format

1. Perceived coding efficiency for the first-round material

For the topics found in the first round of competition, students reported an extremely high perceived coding efficiency (Figure 42). This was anticipated as the first round material is generally basic programming, and most of the topics can be found in the gymnasium and lyceum programming curricula. There was no threshold concept identified within the first-round material. Thus, the scores on tasks associated with the topics were exceptionally high. For the first-round tasks on the Michanicos platform, COI students submitted 505 correct solutions with 1754 total submissions (29%) (Section 5.7 - Table 9).

The average number of users that correctly solved these tasks was 21.5, and the average number of users that attempted the tasks was 24.7. Therefore, the success ratio was 87%. These are statistics from the platform regarding the tasks that appeared in the first round of competition in the past, and they are associated with phase one material.

These results were retrieved in August 2019. A correct submission has been awarded 100 points. Submissions that scored less than 100 points are not accumulated in the correct submissions total.

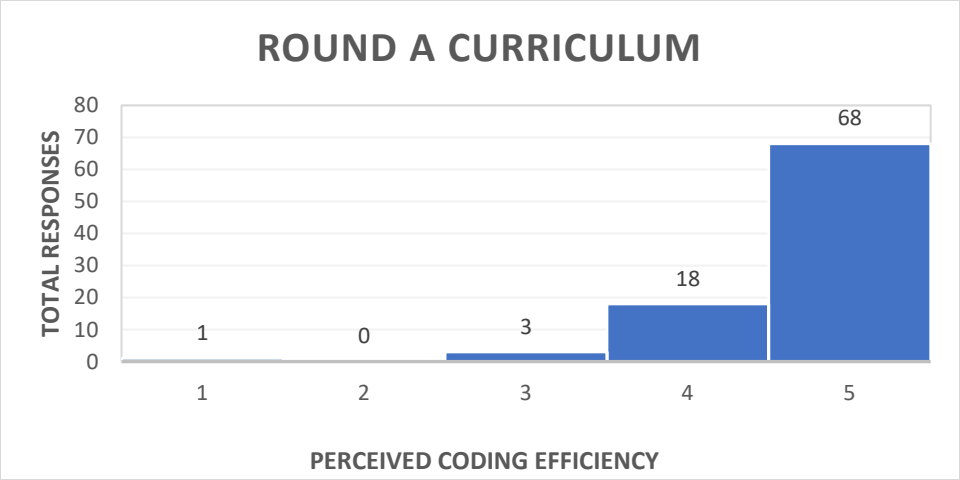


Figure 42: First round perceived coding efficiency

2. Perceived coding efficiency for the second-round material

For the topics found in the second round of competition, students reported a relatively high coding efficiency (Figure 43) even though there were two identified threshold concepts: dynamic programming and recursion. However, the DP tasks included in the second round of competition are some of the classic DP problems such as coin change, longest increasing subsequence and knapsack; therefore, solving them was not extremely complicated. Students' perceived coding efficiency for the topics found in the second round was compared with their performances on associated tasks on the Michanicos platform. For the second-round tasks, users submitted 72 correct solutions with 404 total submissions (18%). The ratio for the average number of users that solved them to the average number of users that attempted them was 67% (Section 5.7 - Table 10).

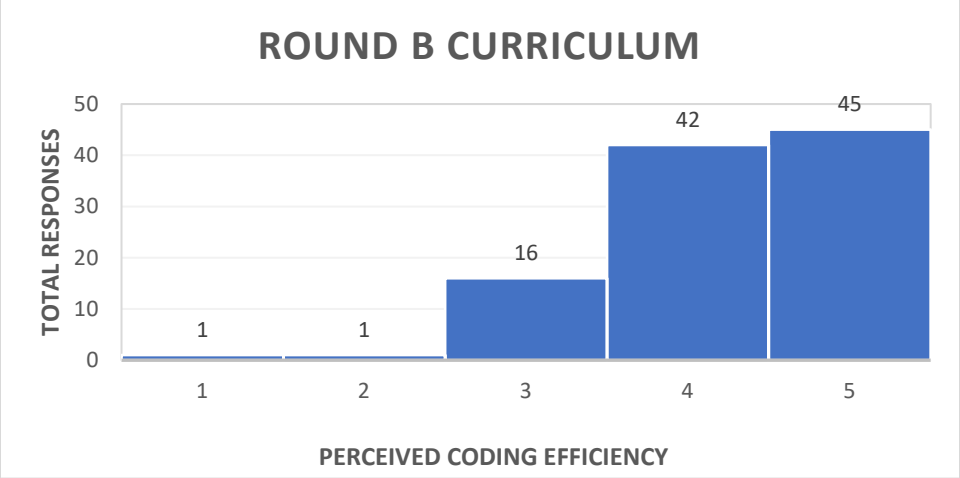


Figure 43: Second round perceived coding efficiency

3. Perceived coding efficiency for the third-round material

The third round traditionally has the most complex tasks, equivalent in complexity to IOI tasks. There were two identified threshold concepts in the topics of this round: dynamic programming optimisations and segment trees. Contrary to the classical DP problems, DP optimisations require special techniques that further augment the complexity of DP solutions. For segment trees, as with the DP optimisations, there is no single approach for solving them either. These concepts were taught during the final lectures and the Easter camp. Most of the students' knowledge was theoretical, with many of them reporting an inability to code the topics efficiently and with confidence. Hence, for the third-round topics, the perceived coding efficiency reported was relatively moderate (Figure 44). For the third-round tasks, users submitted 45 correct solutions with 391 total submissions (12%). The ratio for the average number of users that solved them to the average number of users that attempted them was 59% (Section 5.7 - Table 11).

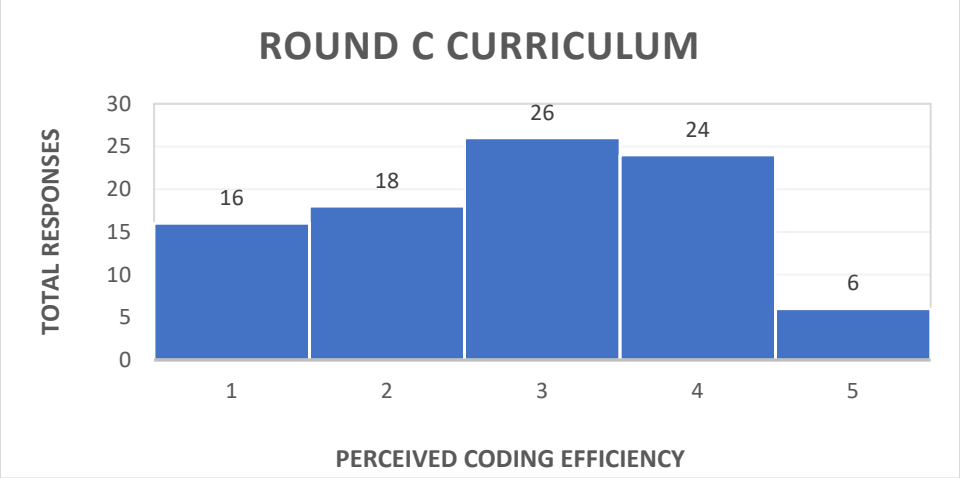


Figure 44: Third round perceived coding efficiency

4. Perceived coding efficiency and actual coding efficiency

I studied the relationship between the students' perceived coding efficiency and their actual understanding of the identified threshold concepts. Specifically, I calculated the students' average scores from ten corresponding programming tasks on the Michanicos platform. I used the scores as a measurement of actual coding efficiency along with self-reported measures of perceived coding efficiency using a Likert scale from 1 to 5 (see Table 6). The perceived coding efficiency level (1-5) was correlated with the students' rank (1-15) based on their average scores rankings. I compared the two datasets using

Kendall's rank correlation (Kendall, 1955). Kendall's Tau, as it is often called from the Greek letter T, was calculated using the Python programming language and, more specifically, the Kendall Tau function from the SciPy package (SciPy, 2019). I have written the programming scripts in Python 3.6.1 that produced the following values for the dynamic programming tasks (M = 63.15, r_t = 0. 810965295015, p-value < 0.001) and for the segment trees tasks (M = 59.88, r_t= 0. 867398958024, p-value < 0.001). The Tau coefficient was .81 for and .87, respectively, indicating a strong relationship between the rankings for both threshold concepts. Interestingly, the perceived level of coding efficiency for the threshold concepts was not always associated with their actual scores on the corresponding tasks, as students have reported underestimating and overestimating their coding capability.

From the analysis of the scatter graphs (Figures 45-46), it is clear that the levels of perceived coding efficiency overlap with each other. Individually, students in the middle level of perceived coding efficiency have acquired scores that span through the lower and upper levels for both threshold concepts identified. The empirical data suggested that there is a discrepancy between students' perceived coding efficiency and their actual understanding of the concepts. The discrepancy can be justified by linking to the partial understandings of threshold concepts reported in Section 5.3. Partial understanding is the time when students attempt to conquer a concept but have not yet succeeded, thus have not yet progressed from an abstract understanding to a robust application and theoretical appreciation of the concept. However, the averages from all of the actual students' scores reported a substantial differentiation of the perceived levels for the dynamic programming associated tasks (L1: no data; L2: M=28.40, N=1; L3: M=49.49, N=7; L4: M=77.6, N=5; L5: M=92.3, N=2) and the segment trees associated tasks (L1: M=18.20, N=1; L2: M=28.60, N=1; L3: M=48.00, N=6; L4: M=76.28, N=5; L5: M=91.10, N=2).

For better illustration of the Likert-scale levels on the scatter graphs, I have distributed the percentages as follows: level 1 (0%-20%), level 2 (21%-40%), level 3 (41%-60%), level 4 (61%-80), level 5 (81%-100%). The distinct colours in the graphs represent each associated programming task and the corresponding performance scores for the distinct levels (Figures 45-46).

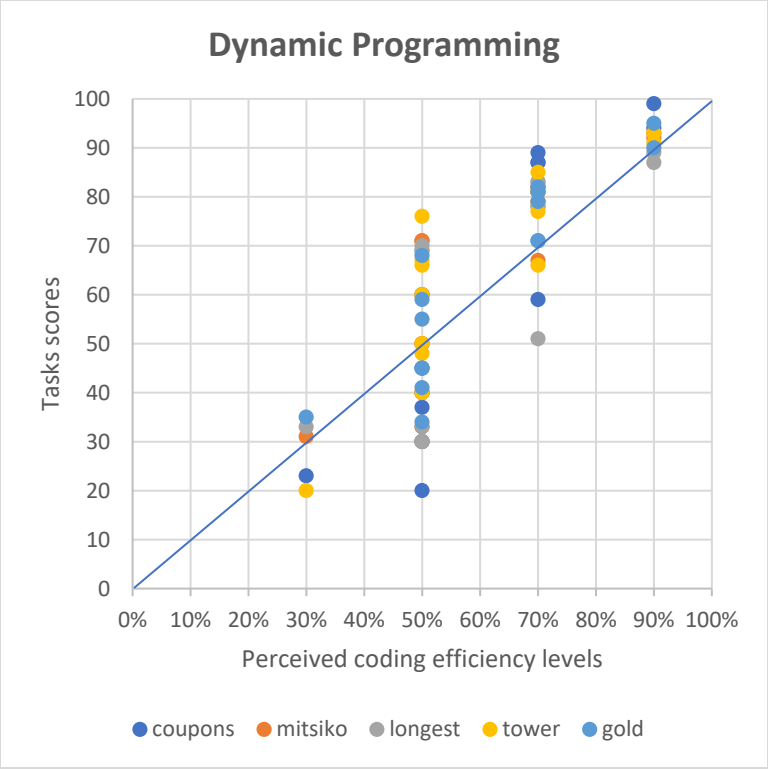


Figure 45: Perceived and actual understanding (Dynamic Programming)

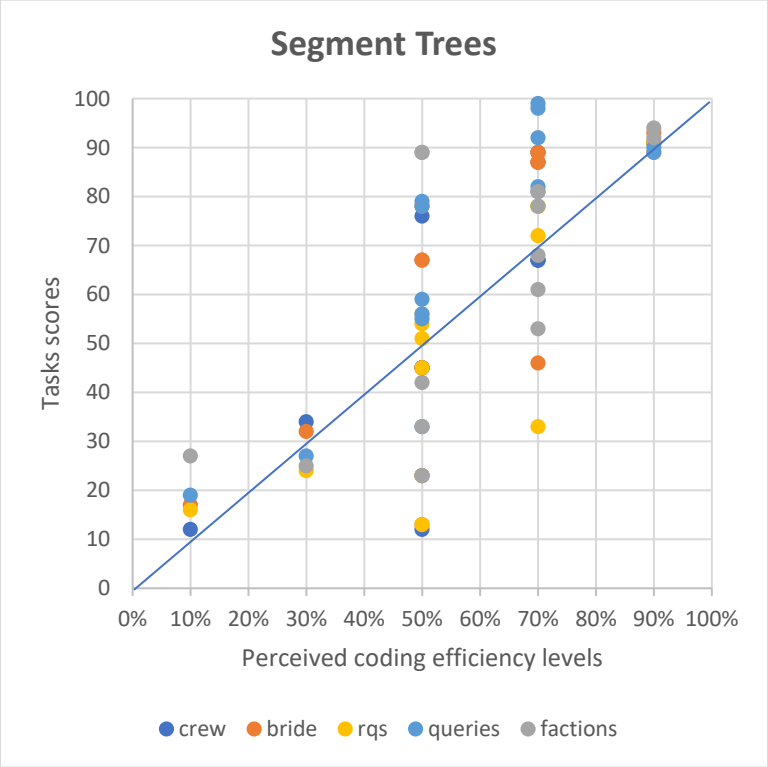


Figure 46: Perceived and actual understanding (Segment Trees)

5. Total solved tasks on the Michanicos platform

For the selected COI students, I made general queries to determine the level of engagement with the Michanicos platform (Figure 47, Table 7). Since the platform was a relatively new introduction to the lectures, the results are based only on the previous year of study. These tasks were not solved in an arbitrary order but instead assigned to the lectures during this research. These are only the solved tasks that awarded 100 points to students for solving them optimally. Any submissions earning fewer points were not included in the statistics.

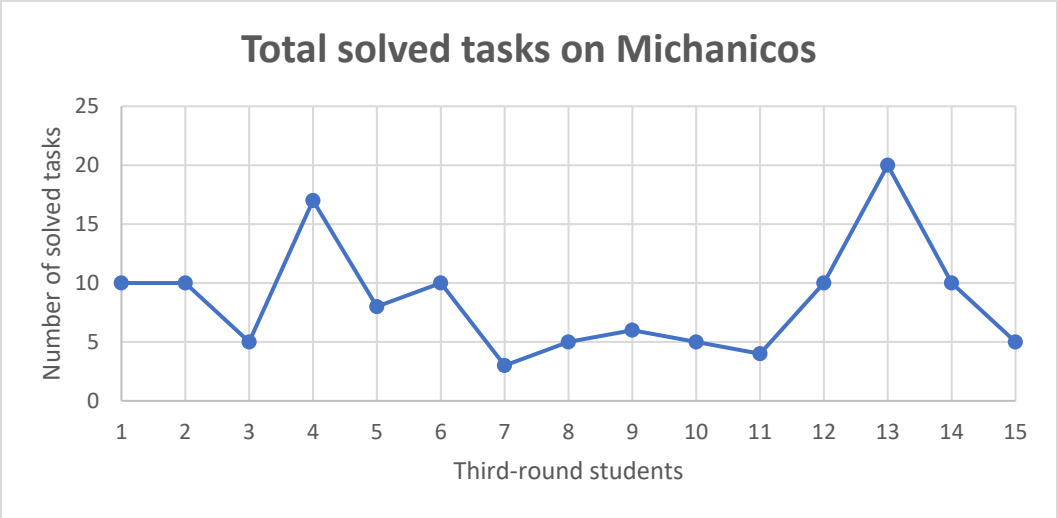


Figure 47: Solved tasks on the Michanicos platform

Total	Average	Standard Deviation	Minimum	Maximum
128	8.53	4.79	3	20

Table 7: Statistics for solved tasks on Michanicos

The total number of solved tasks can be misleading when determining the actual level of student engagement with the platform. For example, a student with only three successful submissions may have more than two hundred submissions that scored fewer points. On the other hand, students who managed to get a full score with only a few submissions will not interact with the same task. They will do so only if they are asked to improve their strategy or the time complexity of their code. Monitoring students' progress through their engagement should, therefore, not be based on just quantitative data. Some qualitative data is required in the form of the students' source code. The qualitative analysis of the source code can help teachers offer improved and controlled advice. The platform offers complete visibility for the students' submissions. Although solving a problem is the

ultimate goal, in some cases, several additional parameters must be considered. User Dremix10 has optimally solved eight tasks on Michanicos but currently has more than one hundred submissions. Working during the summer holidays and gradually improving the scores on tasks is equally a sign of progress, engagement and dedication, which may not always appear in the empirical data (Figure 48).

Time	User	Task	Status	Files	Token	Official
2019-08-06 15:03:28.476188	Dremix10	Trucks	Scored (40.0 / 100.0)	Trucks.cpp	No	Yes
2019-08-06 11:09:58.584546	Dremix10	Trucks	Scored (20.0 / 100.0)	Trucks.cpp	No	Yes
2019-07-24 13:11:21.173886	Dremix10	Infinity	Scored (13.0 / 100.0)	Infinity.cpp	No	Yes
2019-07-24 13:10:41.935299	Dremix10	Infinity	Scored (13.0 / 100.0)	Infinity.cpp	No	Yes
2019-07-24 13:09:28.317349	Dremix10	Infinity	Scored (13.0 / 100.0)	Infinity.cpp	No	Yes
2019-07-24 11:16:05.813443	Dremix10	Infinity	Scored (13.0 / 100.0)	Infinity.cpp	No	Yes
2019-07-22 20:51:49.754845	Dremix10	Infinity	Scored (0.0 / 100.0)	Infinity.cpp	No	Yes
2019-07-22 20:03:05.499376	Dremix10	Infinity	Scored (0.0 / 100.0)	Infinity.cpp	No	Yes
2019-07-22 15:42:55.376463	Dremix10	Infinity	Scored (0.0 / 100.0)	Infinity.cpp	No	Yes
2019-07-19 15:57:01.906857	Dremix10	Game	Scored (50.0 / 100.0)	Game.cpp	No	Yes
2019-07-19 15:56:13.443801	Dremix10	Game	Scored (0.0 / 100.0)	Game.cpp	No	Yes
2019-07-19 15:19:07.887572	Dremix10	Game	Scored (50.0 / 100.0)	Game.cpp	No	Yes
2019-07-19 15:18:24.857557	Dremix10	Game	Scored (50.0 / 100.0)	Game.cpp	No	Yes
2019-07-19 15:18:01.540966	Dremix10	Game	Scored (50.0 / 100.0)	Game.cpp	No	Yes
2019-07-19 15:17:10.051973	Dremix10	Game	Scored (50.0 / 100.0)	Game.cpp	No	Yes
2019-07-19 15:15:12.430016	Dremix10	Game	Scored (50.0 / 100.0)	Game.cpp	No	Yes
2019-07-18 21:28:14.657624	Dremix10	Magic	Scored (30.0 / 100.0)	Magic.cpp	No	Yes
2019-07-17 15:12:32.614489	Dremix10	Six	Scored (20.0 / 100.0)	Six.cpp	No	Yes
2019-07-17 15:06:40.140577	Dremix10	Six	Scored (16.0 / 100.0)	Six.cpp	No	Yes
2019-07-17 14:52:30.864588	Dremix10	Six	Scored (12.0 / 100.0)	Six.cpp	No	Yes
2019-07-17 14:51:59.979912	Dremix10	Six	Scored (4.0 / 100.0)	Six.cpp	No	Yes
2019-07-17 14:50:51.483524	Dremix10	Six	Scored (0.0 / 100.0)	Six.cpp	No	Yes
2019-07-17 14:49:10.758613	Dremix10	Six	Scored (0.0 / 100.0)	Six.cpp	No	Yes
2019-07-17 14:43:12.132201	Dremix10	Six	Scored (8.0 / 100.0)	Six.cpp	No	Yes
2019-07-17 14:32:37.446981	Dremix10	Six	Scored (8.0 / 100.0)	Six.cpp	No	Yes

Figure 48: Submissions by user Dremix10

It is also significant to recognise the students' level of commitment during every week of lectures and throughout the year. The percentage of students attempting to solve two or more tasks per week on the Michanicos platform is measured at 80% (Figure 49). This number is promising as the tasks are not mandatory, and solving them is not explicitly required by the COI students. Instead, it is a voluntary action that determines which students want to perform well and become members of the national delegations. This level of involvement and engagement is linked to the findings from the investigation of the pre-liminal space from Section 5.2.

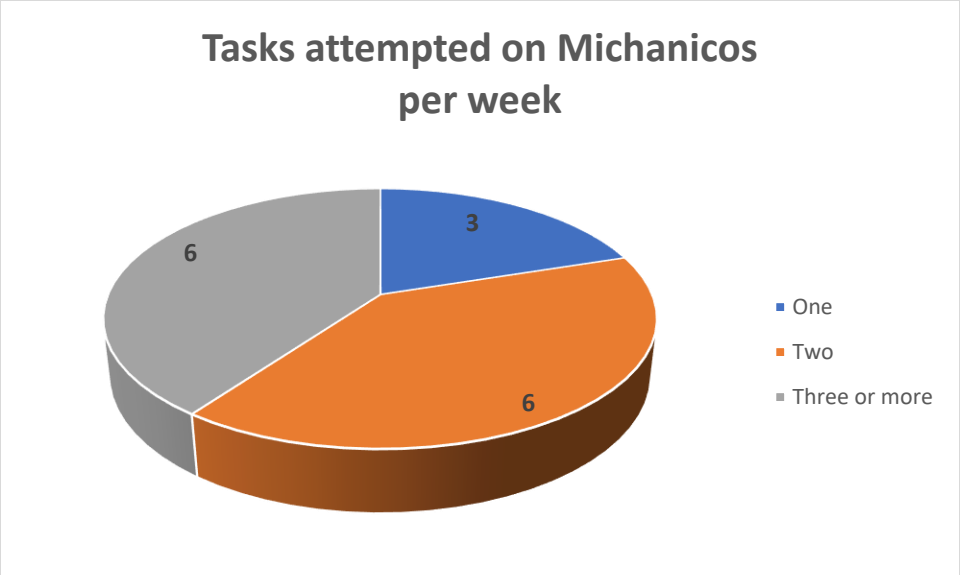


Figure 49: Tasks attempted per week on Michanicos

6. Total solved tasks on other platforms

Since the introduction of the Michanicos platform is relatively recent and since most of the COI students are fluent English speakers, the use of the following online judges was highly recommended: Codeforces, Spoj, UVa, HackerRank, Hellenico and Usaco. These platforms served as extra-curricular material, and students could also choose to participate in online competitions organised by these platforms. The main advantage was that online programming competitions on these platforms were organised daily. Unfortunately, I could not organise daily training contests on Michanicos during this research study because of time constraints and lack of personnel.

COI students who chose to participate in these competitions encountered a much harder international competition than local competitions. Most students have been assigned and solved a relatively large number of tasks on these platforms. Monitoring students' progression on these platforms was based on their ratings and statistics. The numbers shown in Figure 50 and Table 8 are based on the students' responses verified from the rankings on these programming platforms. These statistics are indications of the students' performance on a global stage with unknown programming tasks created by international colleagues from around the world.

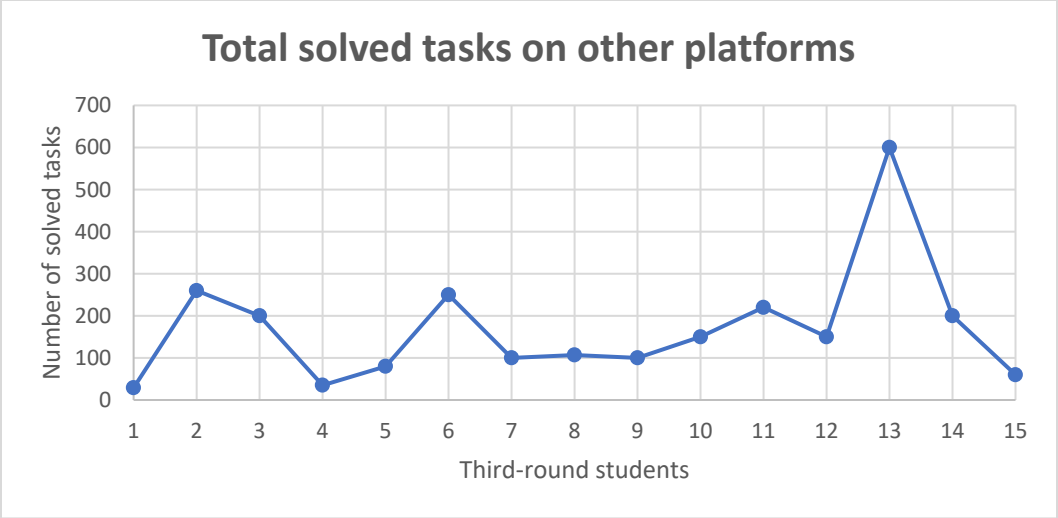


Figure 50: Total solved tasks on other platforms

Total	Average	Standard Deviation	Minimum	Maximum
2541	169.40	140.55	29	600

Table 8: Statistics for solved tasks on other platforms

5.5 Code optimisations/Programming strategies

I found significant empirical data from the students' submissions. I believe that every programming study that wants to improve the students' programming skills and, subsequently, their strategies should focus on the students' source code. The research's findings were not only based on the quantitative scores accumulated by the students individually. The total score is not always a good indication of an improved programming strategy. By examining the source code, teachers can recognise specific coding trends and habits of individual students. Studying code is time-consuming and cannot always be carried out by competitive programming instructors. Nevertheless, it is a critical process as I can provide feedback, correct possible misperceptions and perhaps discover a whole new approach for solving a task. Qualitative code analysis is arguably the best approach for supporting the recruits. At the same time, I would expect the more experienced students to develop methods for developing this aptitude.

There were some cases, even from the first-round tasks, that programme optimisation was required to make the programme code run faster. Time and space complexities are essential for improving the speed and memory requirements of a programme. They are

expressed using the big-O notation[6]. The time complexity of an algorithm (Figure 51) is the most crucial factor that determines if a submission is accepted (AC) or receives a TLE (Time Limit Error). It is realistic for most students to receive TLEs when they are still inexperienced, and they commonly use the brute-force approach as it is the first idea that comes to mind. The brute-force approach generates all possible solutions and determines the correct one. This approach also produces the worst possible time complexity.

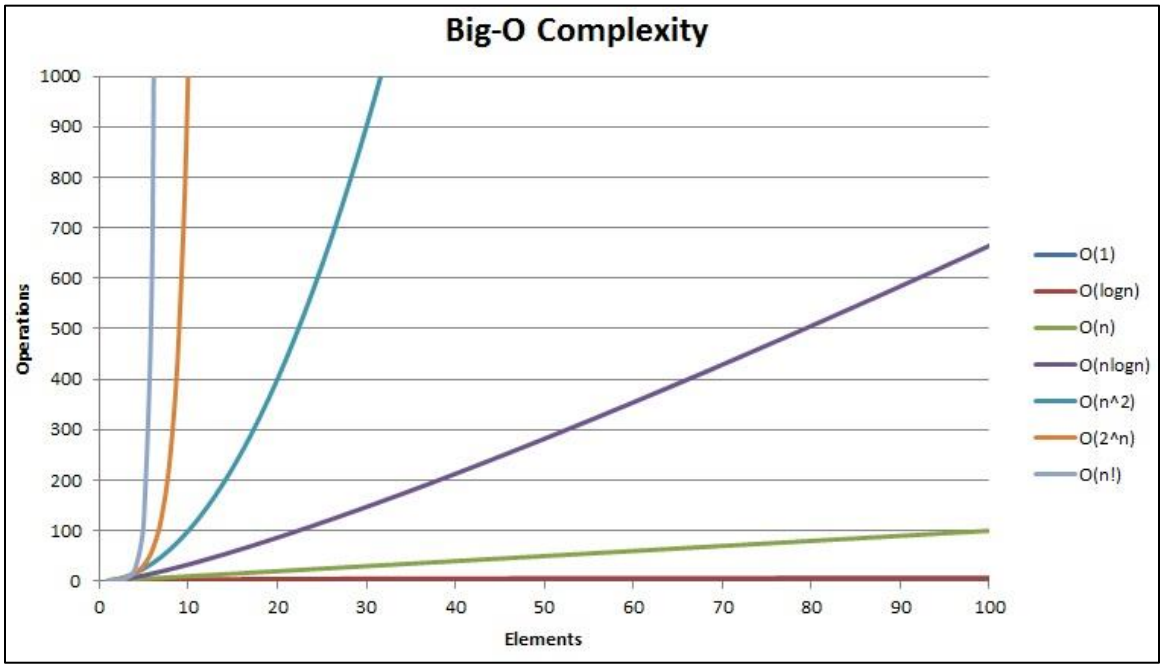


Figure 51: Big-O Complexities from O(1) to O(n!)

To define the students' ability to improve the time complexity of their programmes, the time limit of the task should be twice as much as the optimal solution. As a rule of thumb, we presume modern computers can run 10,000,000 loops in one second. Therefore, for a simple task such as finding a single integer in an array of 10,000,000 distinct integers, a single loop to iterate all of the numbers until the target number is found has a linear complexity of O(N) and can run in under one second. If we want to find all the pairs of integers with difference K within the same array, we can iterate through two nested loops and compare each element with all the other elements. If the maximum input size for the task is 10,000,000 or 10^7, the solution has a quadratic time complexity of $O(N^2)$. The result of 10^{14} dictates that the programme will require several minutes to run, and a TLE is

[6] In Computer Science the Big O notation is used to define the performance of an algorithm.

unavoidable. If the students can improve the performance of their solutions to run in linear time O(N) or even better in logarithmic time O(logN), they can reduce the execution speed of their programmes. The time limit can easily be adjusted through the platform's administrator panel, permitting modifications even between consequent submissions when testing is required.

There were many examples on the platform of students improving the time complexity of their code. I pinpointed these examples and discussed them with the action learning set during the action research cycles. The first example was from the relatively trivial task, 'Coffee': *Find the smallest length between characters 'K' and 'E' that exist in a sequence of characters of length N (1≤N≤1,000,000). If the character 'B' exists in the sequence, the length is 0.*

I set the time limit for this task to one second and included two subtasks. The first subtask for 50 points had several sequences of characters of length $N \leq 1{,}000$, but for the second subtask for 50 points, the length of the sequences was $N \leq 10^6$. Solutions with quadratic time complexity $O(N^2)$ would pass the first subtask and fail on the second subtask. However, solutions with linear time complexity O(N) would pass both subtasks and receive a full score of 100 points. To illustrate this, the following initial submission of student TF[7] received a TLE for the second subtask due to the two nested loops that yield a quadratic time complexity of $O(N^2)$.

```
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int T, N;
    cin >> T;
    string S;
    while (T--){
       cin >> N;
       int dist = N;
       cin >> S;
       for (int i=0; i<N; i++){
           if (S[i]=='B')
               dist = 0;
```

[7] Students' names are not disclosed in this project. Only their initials or usernames on the platform.

```
            else if (S[i]=='E') {
                 for (int j=0; j<N; j++){
                      if (S[j]=='K'){
                              dist = min(dist, abs(i-j));
                      }
                 }
            }
        }
       cout << dist << endl;
    }
 return 0;
}
```

The feedback of the Michanicos platform was clear and revealed the reason for not awarding points for the bottom five test cases. The execution time of the submission for these test cases was above one second. Therefore, only 50 points were awarded (Figure 52).

#	Outcome	Details	Execution time	Memory used
1	Correct	Output is correct	0.000 s	384 KiB
2	Correct	Output is correct	0.000 s	384 KiB
3	Correct	Output is correct	0.000 s	1.13 MiB
4	Correct	Output is correct	0.000 s	2.63 MiB
5	Correct	Output is correct	0.240 s	4 MiB
6	Not correct	Execution timed out	1.028 s	4 MiB
7	Not correct	Execution timed out	1.044 s	4 MiB
8	Not correct	Execution timed out	1.040 s	4 MiB
9	Not correct	Execution timed out	1.024 s	4 MiB
10	Not correct	Execution timed out	1.072 s	4 MiB

Figure 52: Submission feedback for TLE

It took student TF two days to improve the complexity of the code. By the fourth submission, he received a full score with the following submission that runs in linear time O(N). The AC was achieved by keeping track of the previous occurrences of the two characters in question. If the code detected a 'K' character, it calculated the distance from the last occurrence of 'E' and vice-versa. Then the distance was compared to the minimum distance calculated so far, and if it was a new minimum value, the distance was updated. Hence, this is the optimal solution for this task as it is impossible to improve the O(N) complexity. Student TF did not receive any help from his teachers or his peers during the entire time of attempting to solve this task.

```
#include <iostream>
#include <algorithm>
using namespace std;

int main(){
    int T, N;
    cin >> T;
    while (T--){
        string S;
        cin >> N;
        cin >> S;
        int dist = N;
        int lastE = -N, lastK = -N;
        for (int i = 0; i < N; i++){
            if (S[i] == 'B'){
                dist = 0;
                break;
            }
            else if (S[i] == 'E'){
                dist = min(dist, i - lastK);
                lastE = i;
            }
            else if (S[i] == 'K'){
                dist = min(dist, i - lastE);
                lastK = i;
            }
        }
        cout << dist << endl;
    }
 return 0;
}
```

The results were better this time, earning student TF a full score (Figure 53).

#	Outcome	Details	Execution time	Memory used
1	Correct	Output is correct	0.000 s	128 KiB
2	Correct	Output is correct	0.000 s	128 KiB
3	Correct	Output is correct	0.000 s	256 KiB
4	Correct	Output is correct	0.004 s	128 KiB
5	Correct	Output is correct	0.056 s	256 KiB
6	Correct	Output is correct	0.248 s	252 KiB
7	Correct	Output is correct	0.224 s	252 KiB
8	Correct	Output is correct	0.912 s	380 KiB
9	Correct	Output is correct	0.888 s	388 KiB
10	Correct	Output is correct	0.884 s	388 KiB

Figure 53: Receiving a full score on a task

I observed the second example with student AP who was initially struggling with the following task: *Find the sum of the last digits of all the multiples of M in the range [M…N] ($1 \leq M, N \leq 10^{16}$).* This task was particularly tricky because even a solution with linear time complexity O(N) would not pass the largest test cases of 10^{16}. Student AP initially tried to iterate through all possible numbers and find all the multiples of M, then add their last digits and output their sum. Noticeably, the following code by student AP would get a TLE on the last subtask:

```
#include <iostream>
using namespace std;

int main() {
  long long int N, M, ans=0;
  cin >> N >> M;
  for (long long int i=M; i<=N; i++){
      if (i % M == 0)
      ans += i%10;
  }
  cout << ans << endl;
return 0;
}
```

Student AP struggled for a while to get a full score on this task. A hint from a peer helped student AP realise that for every multiple of M, only ten digits were recurring. So, for M=2, the first ten multiples would be 2, 4, 6, 8, 10, 12, 14, 16, 18 and 20; therefore, the last digits of these numbers would produce a sum of 40. Similarly, for M=7, these digits would be 7, 4, 1, 8, 5, 2, 9, 6, 3 and 0, producing a sum of 45. The following image is from the Slack workplace interaction between the two students as student AP was trying to calculate all the sums manually. Initially, he got most of these sums wrong (Figure 54).

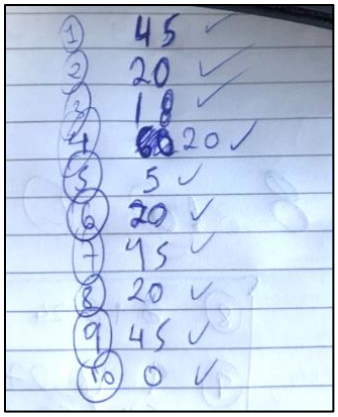


Figure 54: Screenshot from Slack workspace interaction

After storing these digits in an array of ten elements and their sum in a different variable, student AP managed to find the total number of multiples (X = N / M). Once number X was obtained, the following was basic arithmetic. The sum of the first X modulo[8] 10 digits had to be found and added to the original sum multiplied by X / 10. Finally, it was straightforward for student AP to receive a full score with the following C++ code:

```
#include <iostream>
using namespace std;

int main() {

 long long int N, M, X, sum=0, ans=0, temp;
 int arr[10];
 cin >> N >> M;
 X = N / M;
 for (int i=0; i<=9; i++){
    arr[i] = ((i + 1) * M) % 10;
    sum += arr[i];
 }
 temp = X%10;
 for (int j=0; j<temp; j++)
     ans += arr[j];
 cout << ans + (X/10 * sum);

return 0;
}
```

In addition to the time complexity of programmes, there has been evidence of students optimising the space complexity of their code. Space (or memory) complexity is a measurement of the storage capacity a programme requires to run. Generally, in most programming tasks in competitive settings, the memory limits are set higher than time limits, but they are not infinite. In C++, it is not permitted to define a one-dimensional array larger than $\sim 10^8$ of 32-bit integers. The storage requirements are proportional to the number of elements in the array or any other data structure used in the programme. Therefore, we can easily calculate the memory requirements of the programme. The integer data type requires 4 bytes of memory, and the declaration of an array of 1,000,000 integers is made as follows:

```
int arr[1000000];        // declaring an array of integers
```

[8] Finds the remainder for a division of two numbers.

For the above statement, the total requirement will be 4 x 1,000,000 bytes or 3,906.25 kilobytes of memory. When the memory requirements increase linearly with the increase in the size of the array (N), we use the big-O notation as with the time complexity. Therefore, we have linear space complexity symbolised as O(N). Students are encouraged to write code efficiently to keep the space complexity to a minimum.

The third example is from a dynamic programming task called 'Paths': *Find the number of possible paths between points A and B, moving only to the right or down from the current position, on a 2-D grid of dimensions N x M* $(2 \leq N, M \leq 1{,}000)$. For better illustration, I included the following image in the task description (Figure 55).

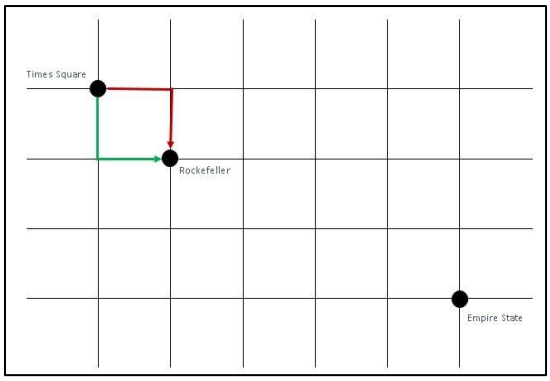


Figure 55: How many possible paths exist from Times Square to the Empire State building?

For solving this task, students initially made one significant observation. To be able to reach any other point on the same line going either right or down, there is only one way to do so. To find the total paths to reach the point (i, j), we must add the paths from the previous two points. Accordingly, if we want to reach point $P[i, j]$, then we must calculate the points $P[i, j-1]$ and $P[i-1, j]$ as this is the total number of paths to get to point $P[i, j]$.

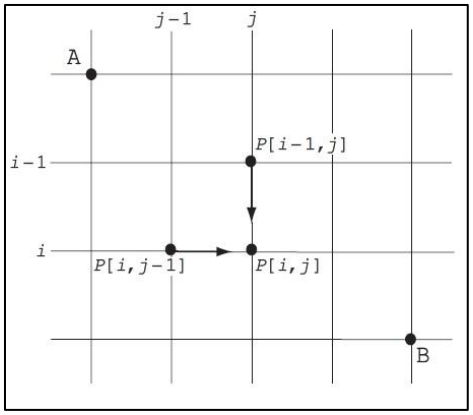


Figure 56: Calculating paths to reach point P

Most students were eager to use a two-dimensional array to solve this task. They used the following equation to fill the first row and the first column with ones (1), indicating that there was only one path to reach these points.

1. $P[0, j] = 1$ for every $0 \leq j \leq M\text{-}1$ and $P[i, 0] = 1$ for every $0 \leq i \leq N\text{-}1$

Then they used the following equation for filling the remaining squares of the 2-D array.

2. $P[i, j] = P[i - 1, j] + P[i, j - 1]$ for every $1 < i \leq N\text{-}1, 1 < j \leq M\text{-}1$

The following was the most common solution produced using a 2-D array:

```
#include <iostream>
using namespace std;

long long int paths[1000][1000];

int main() {
 int N,M;
 cin >> N >> M;

 for (int i=0; i<N; i++){
    for (int j=0; j<M; j++){
       if ((i==0) || (j==0))
            paths[i][j]=1;
       else
            paths[i][j] = (paths[i-1][j] + paths[i][j-1]);
        }
      }
    cout << paths[N-1][M-1] % 1000007<< endl;
    return 0;
}
```

A full score was awarded for the solution above, but a closer look by some students, as in the case of student MP, produced a much better space complexity approach. A 2-D array of 1000 x 1000 of long-long[9] integer type requires 7,812.5 kilobytes of memory. For the solution above, students managed to get a time complexity of O(N x M) and a space complexity of also O(N x M) since a 2-D array was used.

When we discussed her solution, MP mentioned that she noticed that in the second equation to calculate $paths[i][j]$, the only values from the entire 2-D array needed were $paths[i\text{-}1][j]$ and $paths[i][j\text{-}1]$. She concluded that she needed only the results from the previous row. Therefore, she switched her 2-D array to a 1-D array and stored the intermediate values of $paths[i-1]$ only. Though the time complexity of student MP's

[9] The 'long long int' data type allows values in the range -(2^{63}) to (2^{63}).

programme remained unchanged O(N x M) due to the nested loops, the space complexity was improved to O(M). The following is MP's solution using a 1-D array, obtaining a space complexity of O(M):

```
#include <iostream>
using namespace std;

unsigned long long int paths[1000];

int main() {
 long long int N,M;
 cin >> N >> M;
 for (int i=0; i<M; i++)
     paths[i] = 1;

 for (long long int i=1; i<N; i++)
     for (long long int j=1; j<M; j++)
            paths[j] += paths[j-1];

    cout << paths[M-1] % 1000007<< endl;

 return 0;
}
```

In the fourth example, there was a similar approach by student CH. He was able to reduce the space complexity of his programme to O(2 x N) from O(N^2) by accumulating the total needed using a similar dynamic programming technique.

Find the number of ways to remove as many characters as possible (0 or more) from a string of characters of size N (1≤N≤20,000) so that the remaining string would be a palindrome[10].

A 2-D array was initially used to store the calculated intermediate values, but the space complexity of O(N^2) resulted in the following outcome triggering memory limit violations:

#	Outcome	Details	Execution time	Memory used
8	Not correct	Execution killed with signal 11 (could be triggered by violating memory limits)	0.000 s	256 KiB
9	Not correct	Execution killed with signal 11 (could be triggered by violating memory limits)	0.004 s	128 KiB
10	Not correct	Execution killed with signal 11 (could be triggered by violating memory limits)	0.000 s	132 KiB
11	Not correct	Execution killed with signal 11 (could be triggered by violating memory limits)	0.016 s	892 KiB

Figure 57: Memory limit exceeded

[10] A sequence of characters which reads the same forwards and backwards.

The observation of student CH enabled him to reduce the required 2-D array to only two rows. He used a modulo operation to keep the intermediate values within the specific constraints. Here is the submission of student CH that received a full score:

```
#include <iostream>
#define M 20130401
using namespace std;
int C[2][20000];

int main() {
 int N;
 string st;
 cin >> N;
 cin >> st;
 for (int i=N-1; i>=0; i--){
   C[i%2][i]=1;
   for (int j=i+1; j<N; j++)
    if (st[i]==st[j])
       C[i%2][j] = (1 + C[(i+1)%2][j] + C[i%2][j-1]) % M;
    else
       C[i%2][j] = (C[(i+1)%2][j] + C[i%2][j-1] - C[(i+1)%2][j-1]) % M;
 }
 cout << C[0][N-1] << endl;
return 0;
}
```

The above qualitative findings signify only a small percentage of the total number of proven code optimisations that I was able to identify and analyse through the platform. I used these optimisations with the action learning set for identifying the programming strategies involved, engagement in the learning process and verification of the teachers' meaningful and substantial support towards the students.

5.6 Student feedback on the framework and its components

In this section, I have reviewed the feedback on the framework's components from the COI students. As reported from the threshold concepts investigation, concrete learning is frequently troublesome and uncomfortable as it requires the students to improve upon their current ways of thinking. Nevertheless, the uncomfortable student in this situation will probably make more progress than the comfortable student. Undoubtedly, students' positive feedback and satisfaction with training methods cannot always be harmonised with their learning progress and academic achievements. Moreover, students' assessment of their current knowledge has been occasionally reported to be undependable (Kruger and Dunning, 1999).

As long as we acknowledge these issues, student feedback can be a valuable resource for educators. Students' views can help teachers identify the weaknesses and strengths of a pedagogical tool and propose enhancements. The engaging factors are significant as they affect student motivation. Consequently, if the students like the framework, it does not necessarily prove its efficiency. Similarly, if they do not like it, it does not prove that the framework is not working. What is imperative is to discover and reflect on the rationale for the students' feedback.

Overall, the Michanicos platform received high praise from almost all students and teachers. In the questionnaire, students were requested to elaborate on the positives and negatives of Michanicos and recommend upgrades. Some of the advantages reported were the submissions' evaluation speed and the interface's simplicity and ease of use. The most critical feature reported was the platform's capability to be used as a personal code repository/portfolio for all the students' solutions. Moreover, the speed of the evaluation process and the appropriate selection of tasks were also mentioned (Figure 58).

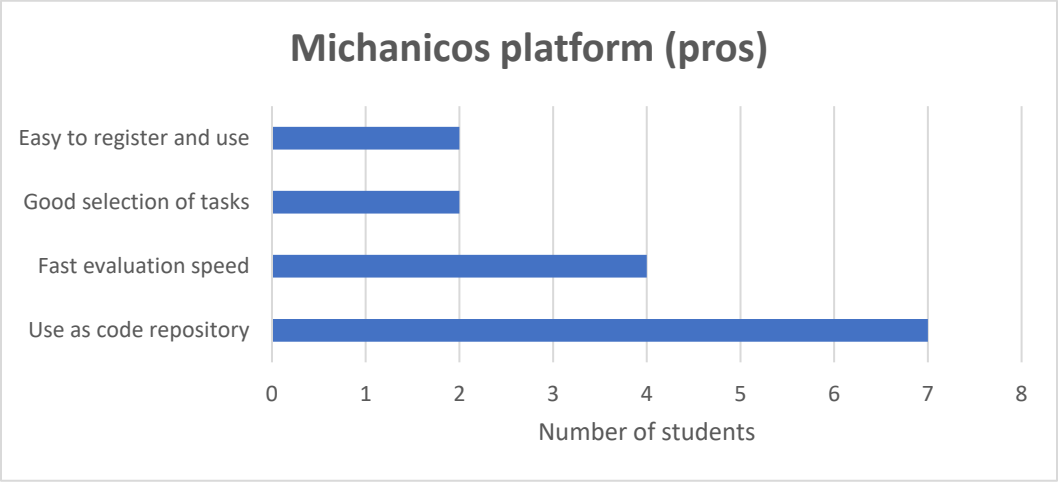


Figure 58: Positive aspects of the Michanicos platform

On the negative side, the students mentioned that they would like to see more tasks associated with each tag and a more extensive selection of tags (Figure 59). As for future upgrades, the students requested a forum for comments and discussions on tasks, availability of test cases for downloading and editorials for the more complex tasks after a solution is reached. Most of the upgrades have been confirmed by the action learning set, and the upcoming version of Michanicos will include the majority of these (Figure 60).

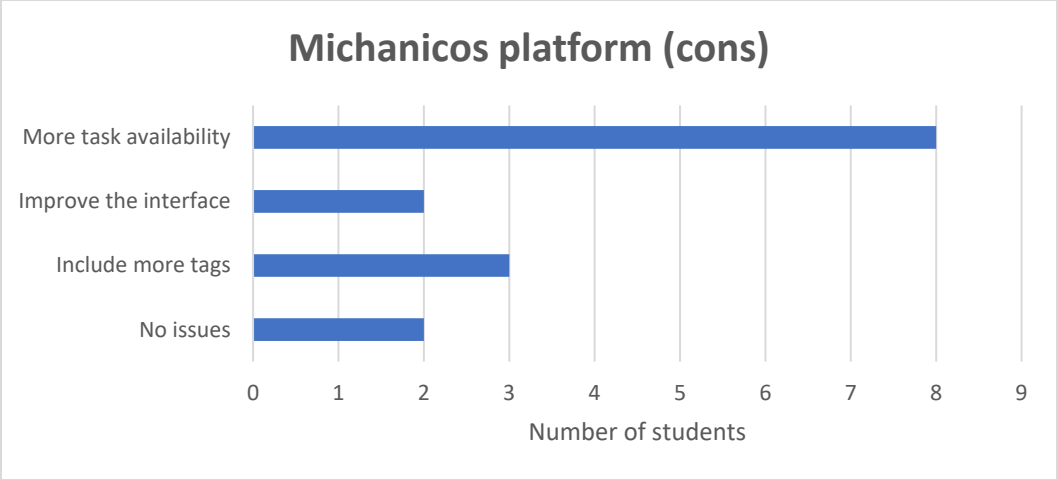


Figure 59: Negative aspects of the Michanicos platform

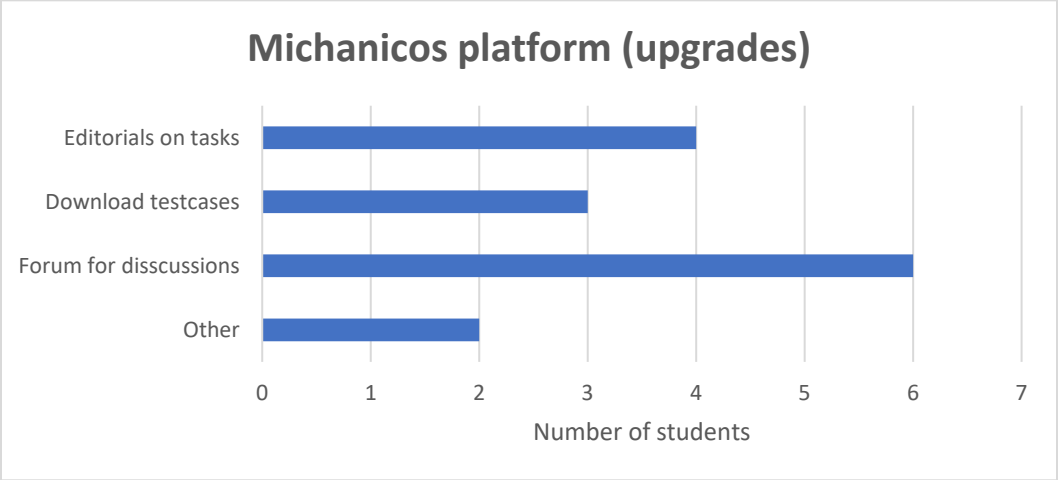
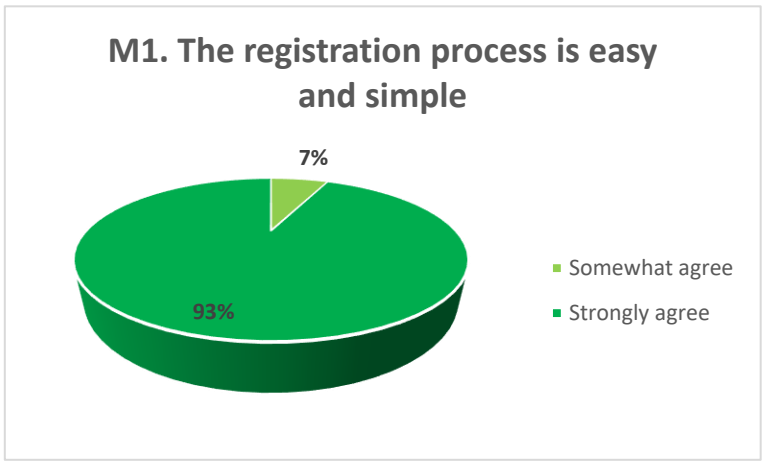
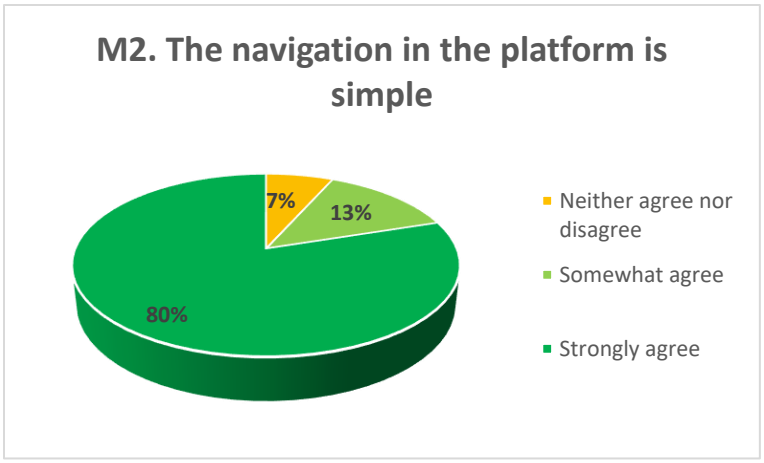
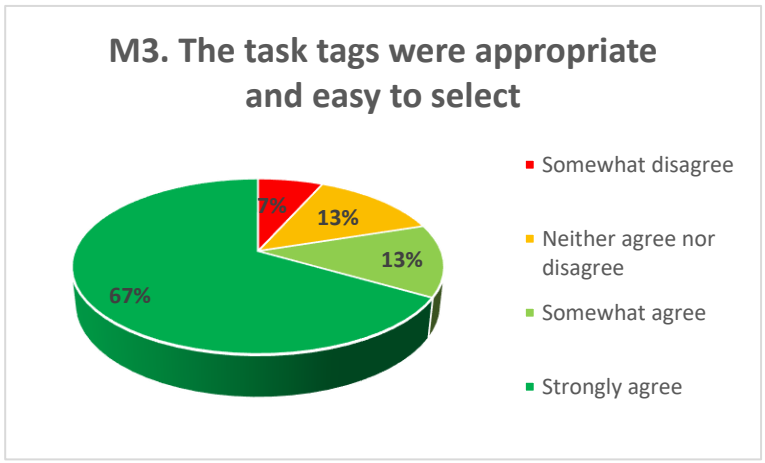


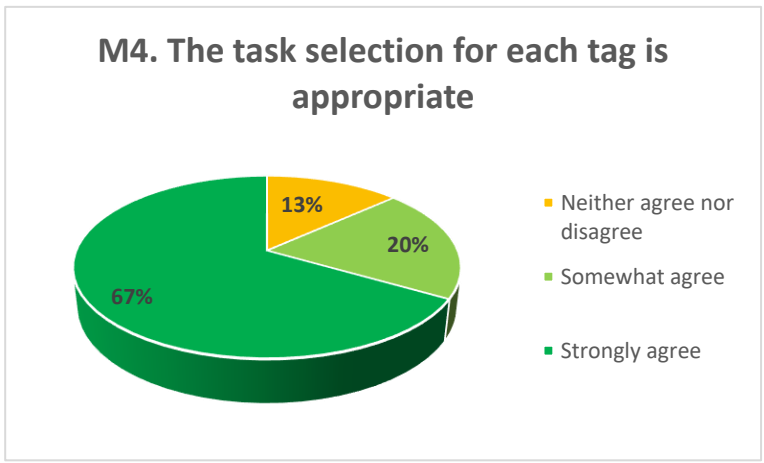
Figure 60: Requests for future upgrades

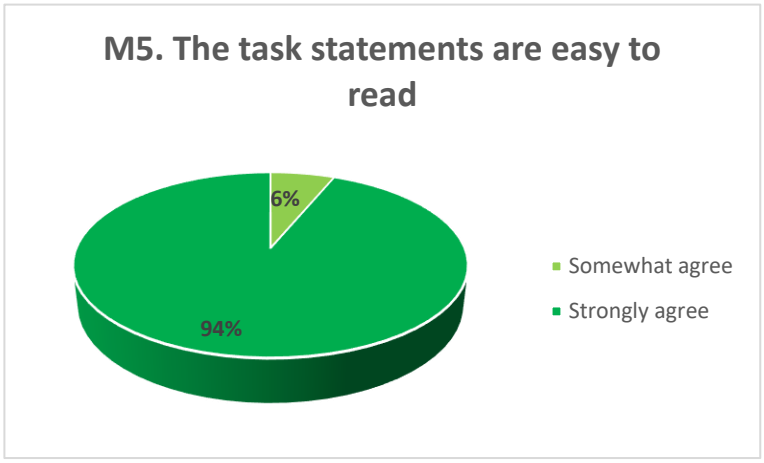
The open-ended feedback of students highlighted the strengths, weaknesses and enhancements of the platform. Furthermore, I collected student responses concerning their interactions with the Michanicos platform using a 5-point Likert scale on the same questionnaire. The action learning set wanted to have extensive feedback on students' perspectives about the platform's usability and simplicity, the suitability and clarity of included tasks and lecture notes, the response time of the evaluation process, the leaderboard component, the tasks' statistics and the students' level of improvement (Figure 61).

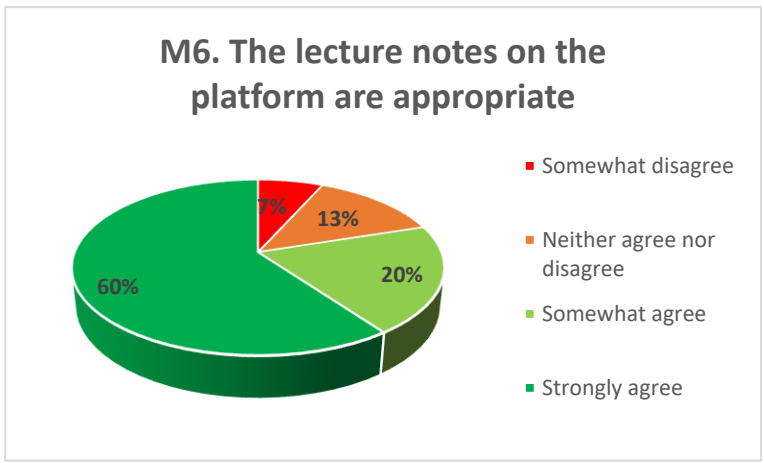

M1. The registration process is easy and simple
7%
93%
Somewhat agree
Strongly agree

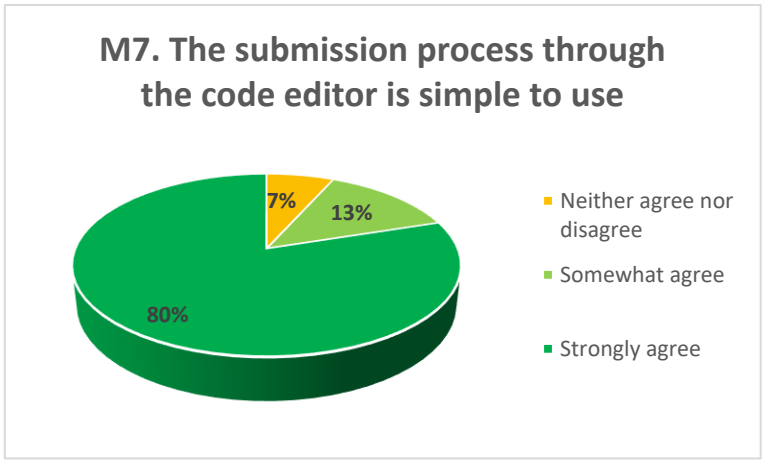

M2. The navigation in the platform is simple
7%
13%
80%
Neither agree nor disagree
Somewhat agree
Strongly agree

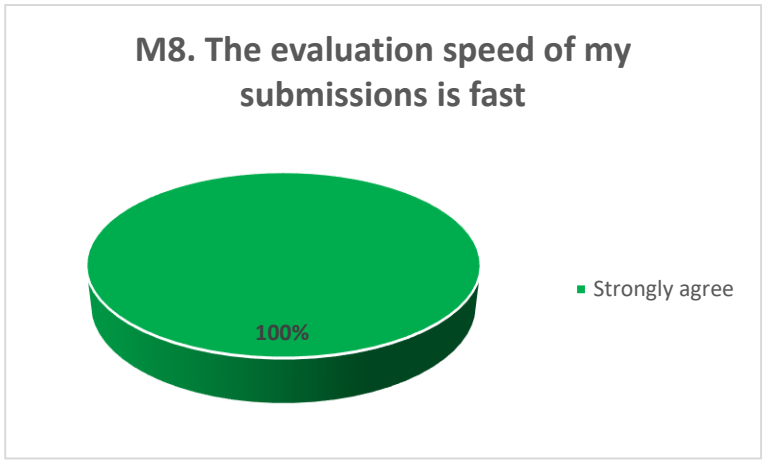

M3. The task tags were appropriate and easy to select
7%
13%
13%
67%
Somewhat disagree
Neither agree nor disagree
Somewhat agree
Strongly agree

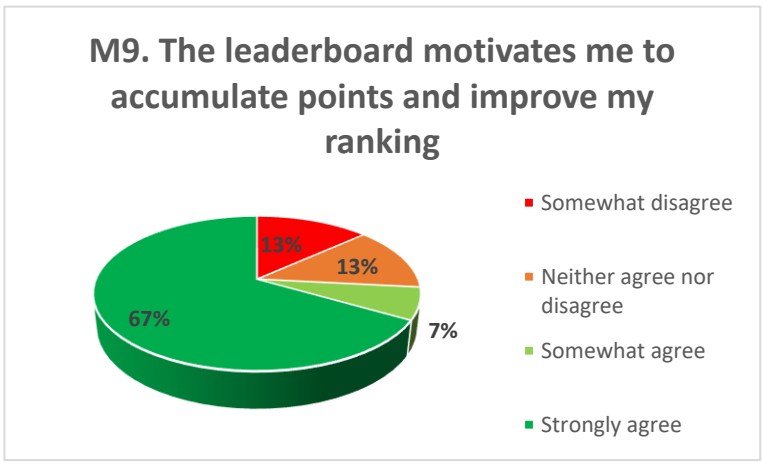

M4. The task selection for each tag is appropriate
13%
20%
67%
Neither agree nor disagree
Somewhat agree
Strongly agree

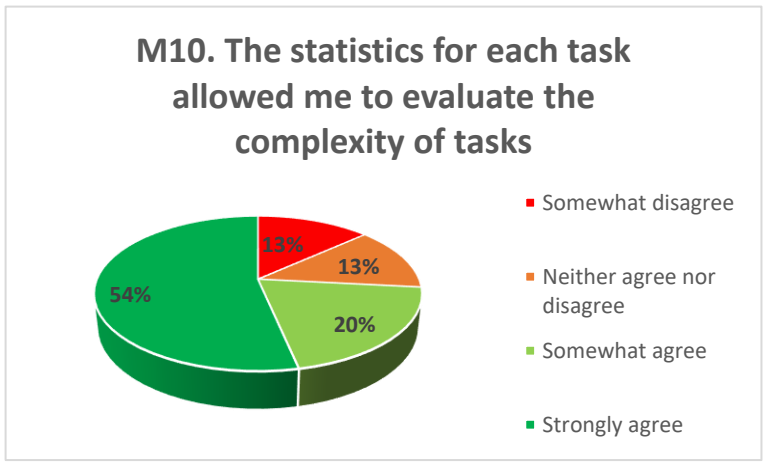

M5. The task statements are easy to read
6%
94%
Somewhat agree
Strongly agree

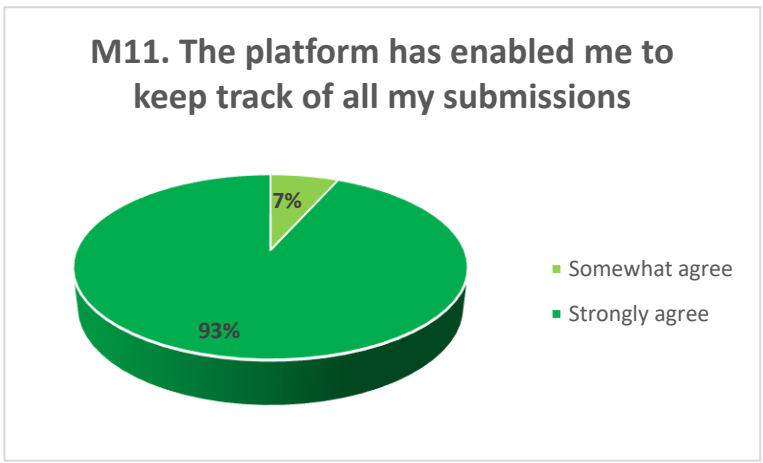

M6. The lecture notes on the platform are appropriate
7%
13%
20%
60%
Somewhat disagree
Neither agree nor disagree
Somewhat agree
Strongly agree











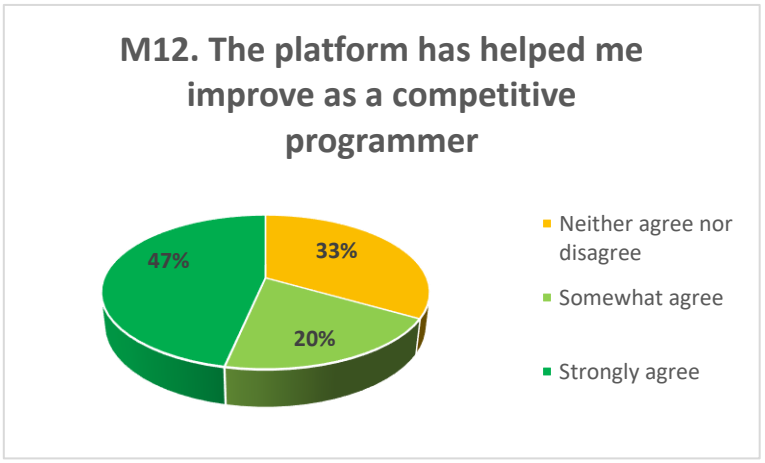


Figure 61: Student feedback on the Michanicos platform

The feedback from the students for the Michanicos platform was mainly positive, as most students had the opportunity to use similar online judges in the past. Hence, any initial concerns about the platform's integration were quickly dismissed. Based on the responses, the platform has established itself as an invaluable component within a year.

The other framework components were also under review within the same questionnaire. The CMS system received very positive feedback, with the most important feature (53%) being the fact that it is the same one used in international programming competitions, and the learning curve was minimal (34%) (Figure 62). As for the COI website that contains all of the material and links for the CMS and the Michanicos platform, students requested more implementations of well-known algorithms within the lecture notes. On the question asking for suggested changes in the training process, 47% of students answered with 'Nothing' and 40% requested 'More material' (Figure 63).

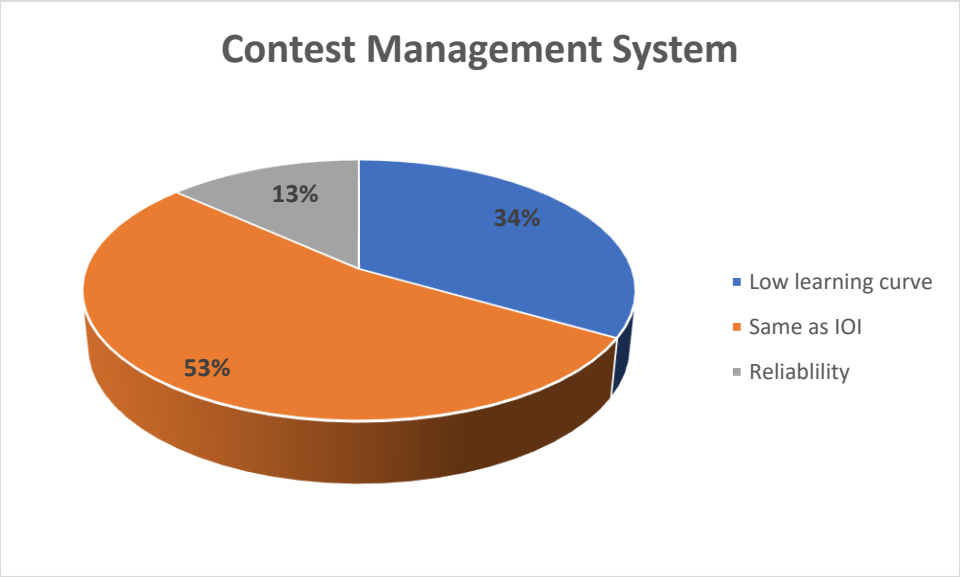


Figure 62: Reviews for the CMS

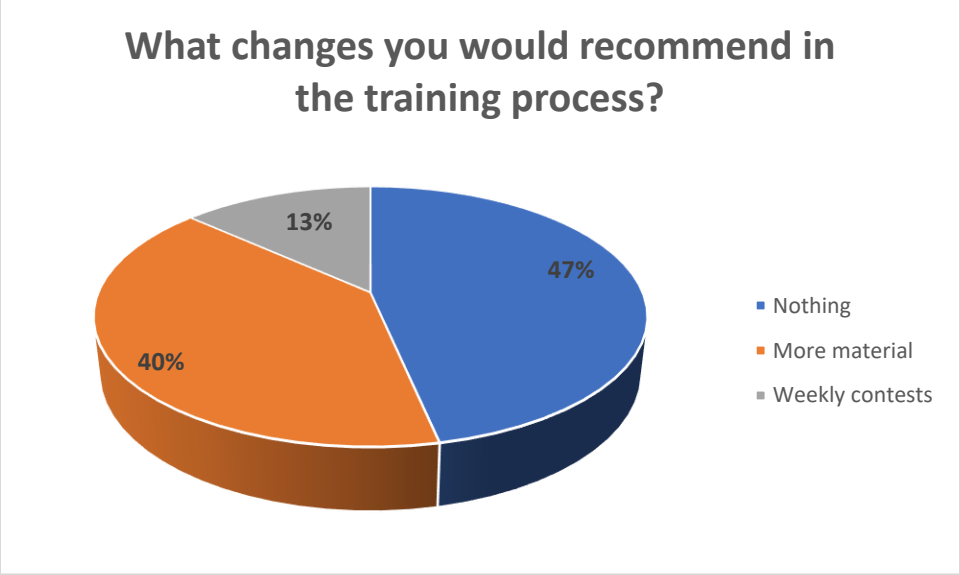


Figure 63: Recommended changes in the training process

5.7 Student engagement with the Michanicos platform

Michanicos platform statistics (August 2018 - August 2019)

Total users: **164**
Total submissions: **3615** (Updated in August 2021: Users **342**, Submissions **10867)**

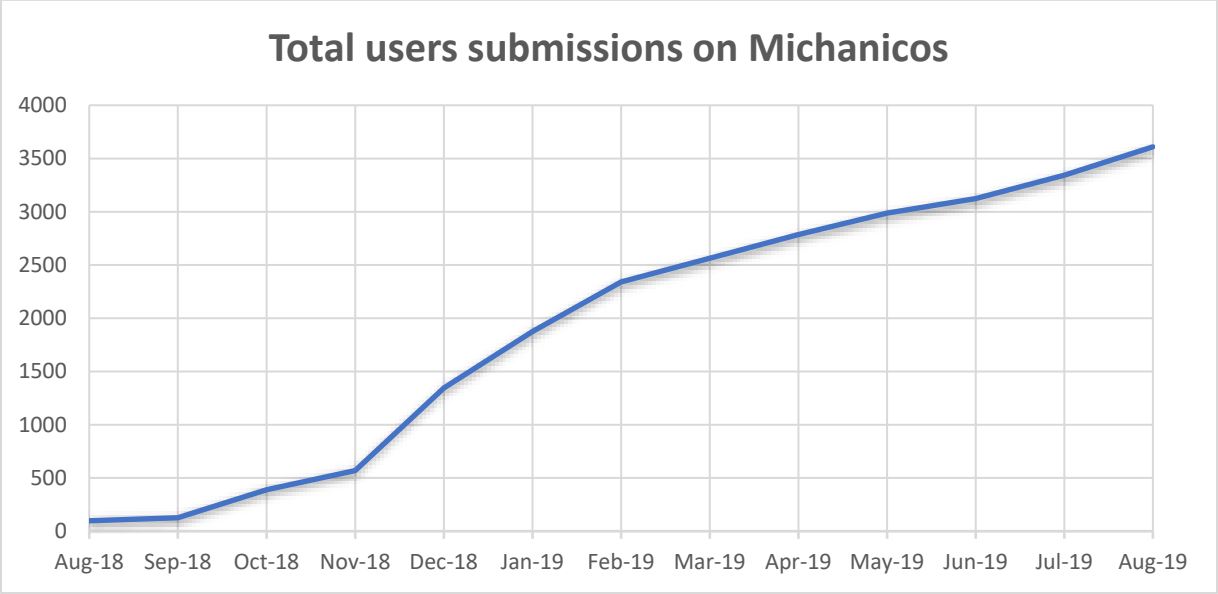


Figure 64: Total user submissions on Michanicos

The level of student engagement with the platform has been steadily increasing during the past year (Figure 64). Accordingly, the students' interactions have also increased as communication and collaboration within the Slack platform indicated. Inevitably, students engaged in energetic methods of learning regardless of the increased required level of cognitive involvement. Learning competitive programming requires a complex mix of context, cognition, engagement and motivation, problem-solving, training and participation. When these components can be combined successfully, it creates powerful ways of thinking about competitive programming. The benefits from the platform usage have been multiple. Arguably, one of the most significant benefits was the ability to use a different approach with students by enforcing increased complexity with advanced students and a step-by-step approach with newcomers. Interestingly, the two methods can be joined by mixing graded level tasks and the proficient use of scaffolding. The platform served its purpose as it offered a balance between freedom in learning and learning management, appropriate for this specific discipline and context and thus promoted the utmost success.

I have analysed the tasks associated with each of the three rounds of competitions to determine engagement levels (Tables 9-11). The columns are defined as follows:

- Users who solved it: Number of users who submitted at least one correct solution.
- Users who tried it: Number of users who submitted at least one submission.
- Correct submissions: Number of submissions with a full score (100 points).
- Total submissions: Total number of submissions for the specific task.

I evaluated the following tables have in Section 5.4.

Task	Users who solved it	Users who tried it	Correct submissions	Total submissions
Shopping	5	6	6	8
Reloaded	2	3	2	31
Email	3	3	3	4
Painter	5	5	5	6
Mission	6	6	7	17
Coffee	6	6	11	17
Couples	4	5	4	16
Caramels	17	17	19	38
Rules	9	11	10	27
Titanic	16	17	20	41
Think	11	14	14	31
Christmas	61	64	89	258
Numbers	58	67	119	455
Oracle	72	85	122	349
Pebbles	47	62	74	456
TOTALS	**322**	**371**	**505**	**1754**

Table 9: First-round tasks' statistics

Task	Users who solved it	Users who tried it	Correct submissions	Total submissions
Coupons	4	6	4	17
Robbery	3	3	3	5
Virus	6	6	7	11
Boxes	6	8	6	30
Tower	5	8	8	41
Froutopia	7	8	8	26
Followers	6	8	6	11
Magic	7	7	7	9
Metro	7	14	8	61
Travel	5	7	7	51
Money	4	8	4	86
Infinity	4	12	4	56
TOTALS	**64**	**95**	**72**	**404**

Table 10: Second-round tasks' statistics

Task	Users who solved it	Users who tried it	Correct submissions	Total submissions
Terbium	2	3	2	6
Cherry	2	5	2	28
Aokigahara	2	4	2	11
Suffix	5	9	6	139
Gold	4	6	4	12
Organization	2	4	6	34
Factions	2	5	2	14
Fence	5	7	5	37
Flow	1	3	1	6
Art	6	8	6	22
Ducks	4	6	5	57
Bacteria	3	4	4	25
TOTALS	**38**	**64**	**45**	**391**

Table 11: Third-round tasks' statistics

5.8 Statistical Analysis of COI Round A

I assessed student performance data from all of our competition rounds. The data provided significant indications to the action learning set (teachers/alumni) about students' strategies, liminal negotiations and task suitability in correlation with the learning objectives. The data is displayed in tables, and the tasks are analysed separately.

Michanicos link: http://81.4.170.42:8980/training/#/tasks/1?tag=COI2019A

Number of contestants: **89**

Tasks	Average Score	Standard Deviation	Group A (Top 33%)	Group B (Middle 34%)	Group C (Bottom 33%)	Complexity Ratio
A. Shopping	87.08	31.63	100.00	93.00	67.59	1.480
B. Think	47.42	47.48	98.50	40.67	1.55	63.478
C. Titanic	21.80	38.19	55.67	8.67	0.34	161.433
D. Strings	59.51	40.71	91.40	73.73	11.79	7.750
TOTAL	215.79	118.0	345.57	216.07	81.28	**4.252**

Table 12: Statistical analysis of the first round

The first round of competition was a reasonably non-trivial qualifying programming round (Table 12). The complexity ratio[11] reveals that the hardest task was 'Titanic' as group C, the lowest-ranked 33% of contestants, got an average of 0.34 points while group A, the

[11] The average score of the top 33% of student scores, divided by the average score of the bottom 33% of student scores for the specific task. It is a measurement of task complexity and it has been used in our action learning set analysis sessions. A value closer to zero shows that the task did not differentiate between the upper level and lower-level students. The higher the complexity ratio value the better the differentiation but extreme values should be avoided.

top 33%, got an average of 21.80 points. Accordingly, the complexity ratio of task 'Titanic' was 161.433. As expected, the easiest task was 'Shopping' which was the known task from the preliminary problems. Most students had the opportunity to solve it before the contest and test the correctness of their solution. It produced a complexity ratio of 1.480.

Tasks analysis of the first round

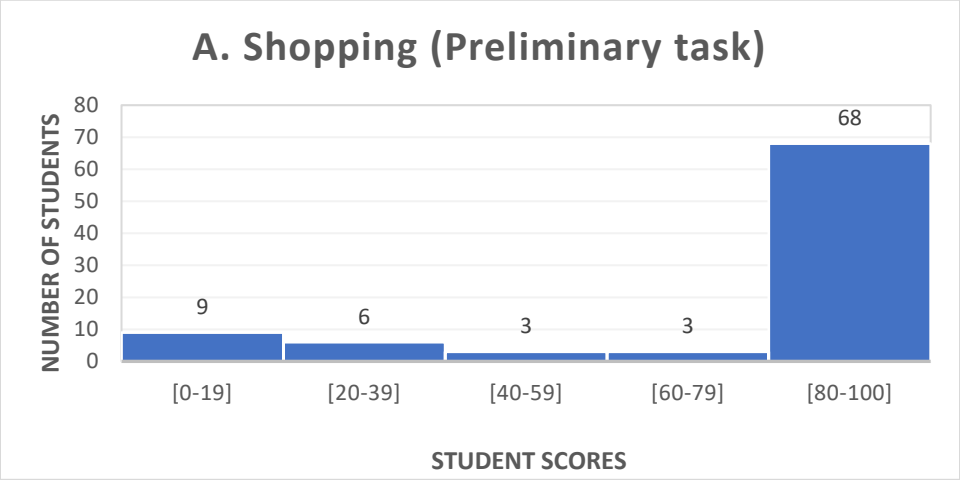


Figure 65: Histogram for task Shopping

Abbreviated task statement: Given two arrays of integers (toys and banknotes), you must calculate the number of toys Benjamin can purchase if the banknote value has to be larger than the value of the next available toy. If not, he has to skip the toy and move to the next one.

The scores for this task produced the highest average score (87.08) as it was a relatively trivial task using 1-D arrays to store the values and calculate the answer iteratively.

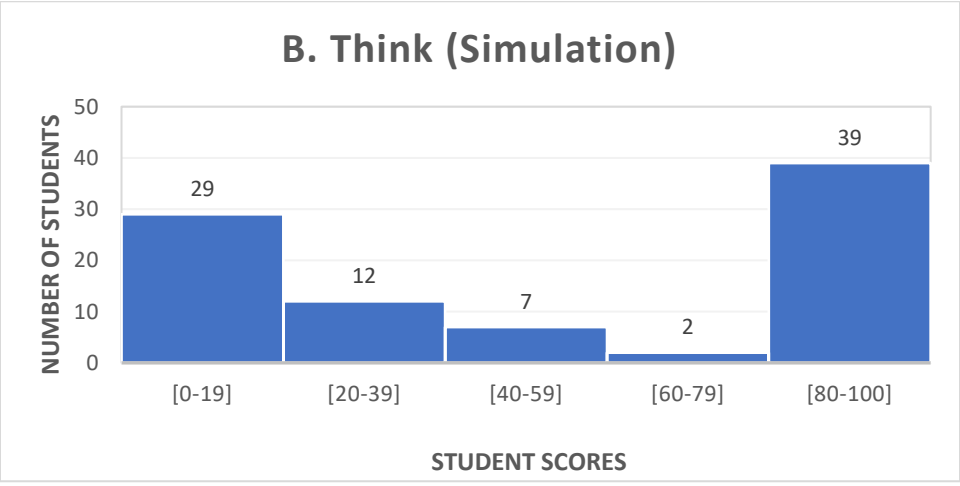


Figure 66: Histogram for task Think

Abbreviated task statement: Given an integer N ($1 \leq N \leq 123,456,789$), print all the digits that, when multiplied by N, produce an anagram of N. For example, if N=1,246,878, the output should be 6 and 7.

According to its complexity ratio, task 'Think' was the second hardest task of the first round. The top 33% of students averaged 98.50 points, while the bottom 33% averaged only 1.55. These results were presumably due to ambiguity in the meaning of an anagram though it was clearly explained in the task statement. Additional remarks from the learning set for some submissions was the difficulty to count the digits of each product and that some submitted solutions had a time complexity of $O(N^2)$ that did not pass the one-second time limit.

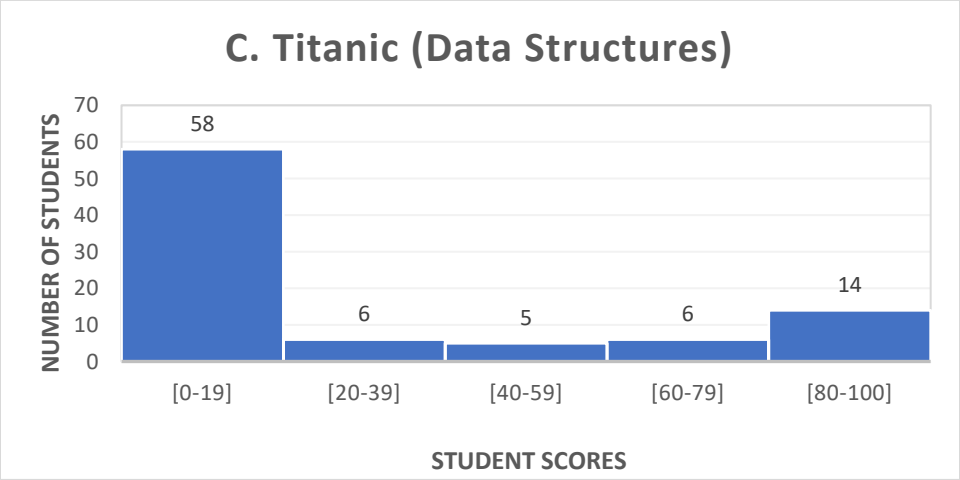


Figure 67: Histogram for task Titanic

Abbreviated task statement: Given the list of passengers and their status, you must output the order of evacuation of Titanic according to the passengers' status.

The number of passengers ($N \leq 1,000,000$), the significantly low time limit (0.9 seconds) and 64 megabytes of the memory limit resulted in the fewest full scores of the contest (12) among all tasks. Task 'Titanic' was the hardest task of the first round based on its complexity ratio of 161.433. The average score from the top 33% was 55.67, which was lower than the average score of the bottom 33% for the task 'Shopping'. The average score from the bottom 33% for task 'Titanic' was a mere 0.34. This average was lower than all tasks, even from the second-round statistics.

I included this task to serve as a tiebreaker. The considerable input size could not be sorted with $O(N^2)$ algorithms such as bubble sort or insertion sort. Additionally, space requirements were tested as submissions that used only two 1-D arrays to store the names and statuses were able to pass. Solutions that included 2-D arrays and multiple arrays for storing each status could not pass the memory limit. In general, this task has served its purpose. Though its point average of 21.80 would be more appropriate for a third-round task, I reached important conclusions such as the students' inclination to use programme memory in abundance and the struggle of some students to improve the time complexity of their code.

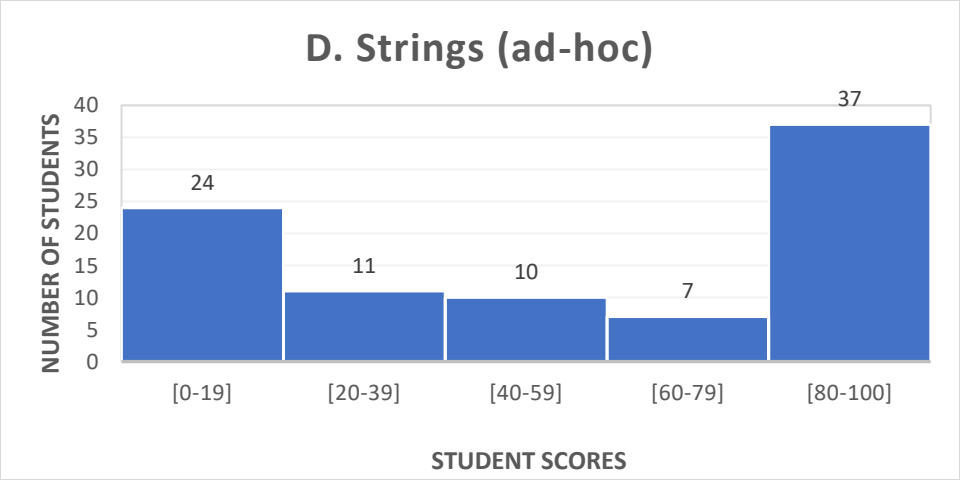


Figure 68: Histogram for task Strings

Abbreviated task statement: You are given a string of characters containing only the characters A, B and C and a set of three rules:

1. You can swap the consecutive characters AB to BA and vice versa.
2. You can swap the consecutive characters BC to CB and vice versa.
3. You cannot swap the consecutive characters AC or CA.

Find the smaller string, lexicographically, which can be produced by making an arbitrary number of swaps (even zero).

For this task, I wanted students to demonstrate improved techniques with linear time complexity rather than brute-force algorithms. From the scores, I concluded that there was a proper distribution of points (top 33% - 91.40, middle 34% - 73.73, bottom 33% - 11.79), and the complexity ratio of 7.750 was the most suitable for this round. The indicative complexity ratio decided in the learning set sessions was a value close to 10.

5.9 Statistical Analysis of COI Round B

Michanicos link: http://81.4.170.42:8980/training/#/tasks/1?tag=COI2019B

Number of contestants: **31**

Tasks	Average Score	Standard Deviation	Group A (Top 33%)	Group B (Middle 34%)	Group C (Bottom 33%)	Complexity Ratio
A. Infinity	27.90	40.22	72.00	12.08	1.86	38.769
B. Metro	25.00	39.32	64.60	11.33	1.36	47.600
C. Money	7.10	20.40	19.40	6.50	0.36	54.320
D. Travel	5.10	8.57	10.80	5.08	2.00	5.400
TOTAL	65.10	86.32	166.80	35.00	5.57	**29.939**

Table 13: Statistical analysis of the second round

The second-round competition was comparatively more challenging than the first round (Table 13). The overall complexity ratio spiked from 4.252 in the first round to 29.939 in the second round. For this round, I have included a difficult task (Money) directly associated with one of the identified threshold concepts (dynamic programming). As expected, it resulted in the highest complexity ratio (54.320) and the second to last average score (7.10). These numbers indicate that students with a good grasp of the threshold concept could earn points, while students with partial or limited understanding failed to do so. The learning set linked the scores for this task with the students' perceived coding efficiency from Section 5.4, and the results were consistent.

Tasks analysis of the second round

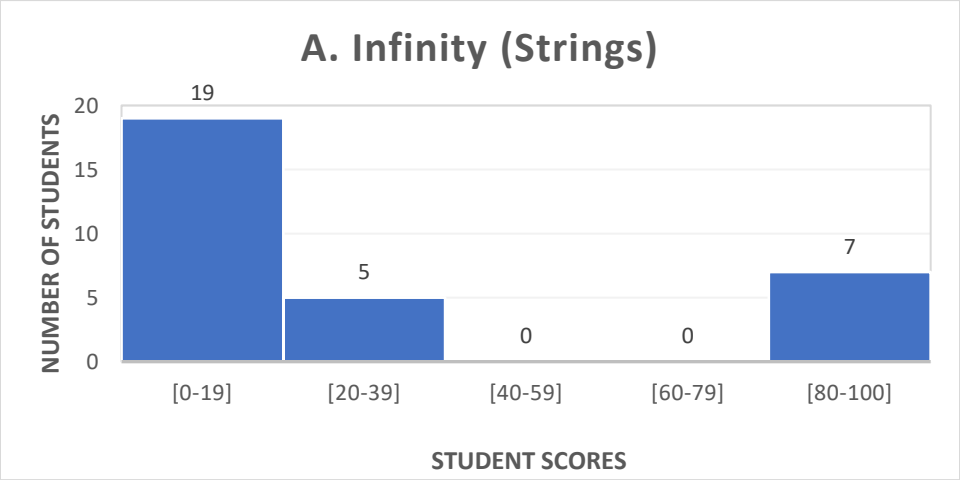


Figure 69: Histogram for task Infinity

Abbreviated task statement: Using the first N ($1 \leq N \leq 26$) characters of the Latin alphabet, find the character in position X of the string created by arranging in increasing order all the possible strings of length one, then all the possible strings of length two and so on. For example, for N=2 (a, b) and X=10, the string should be:

```
a b c aa ab ba bb
```

So, the character at position 10 is b.

I wanted to test the students' ability to produce optimised programming code on a medium difficulty task, so X was a huge number ($0 < X < 10^{18}$). For the first subtask (13 points), the length of the string was not bigger than three characters so that a brute-force solution could pass. I used multiple test cases for the second subtask ($X < 200{,}000, N = 2$) and the third subtask (no constraints). The output should have been multiple characters instead of just one, making it impossible to guess the correct answer. For solving this task, students had to calculate the length of the string in which the X_{th} character was found. So, length L was calculated as follows $L = X - i * i^N$. Then, integer Y was obtained by finding X modulo L. Finally, finding the $(L - Y)_{th}$ digit of number X in base N and adding 'a' would get the character in question.

The task was the easiest of the second round, with a complexity ratio of 38.769 and a relatively low point average (27.90). The top 33% of students got an average of 72.00, which ranked highest in the competition.

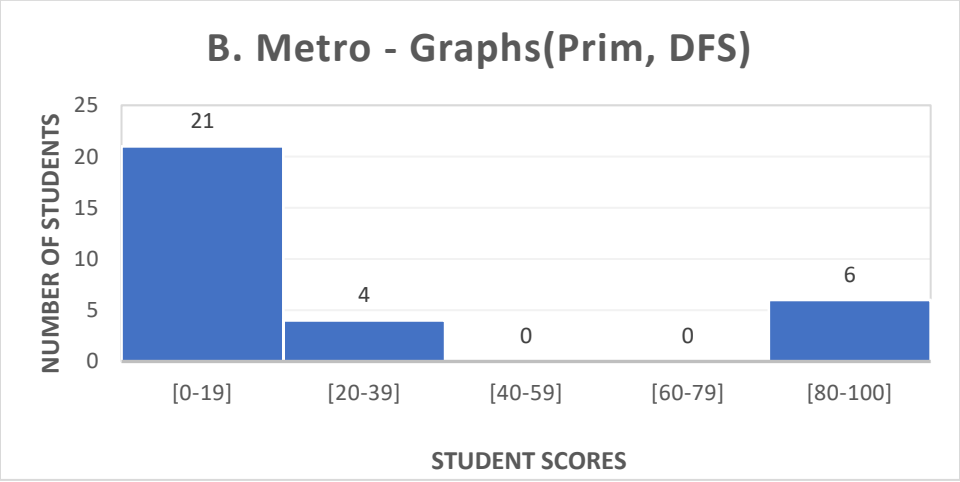


Figure 70: Histogram for task Metro

Abbreviated task statement: Given N cities and M roads, you need to connect all the cities with metro lines, so the total length of these lines is minimal. Then, you need to answer several queries for finding the shortest distance between two cities.

Although task 'Metro' was a pretty straightforward task, it produced the highest complexity ratio of the contest (54.320). Initially, students had to construct the Minimum Spanning Tree[12] (MST) using Prim's or Kruskal's algorithms and then run a DFS (Depth-First Search) algorithm to determine the shortest distances among cities. Task 'Metro' could have been approached in many ways. Between the task creator and testers, there were four different solutions, each one scoring a different number of points.

The learning set determined that several students struggled to store the MST in a proper data structure. Others created the MST but then attempted to find the minimum distances using Floyd-Warshall and Dijkstra's algorithms which did not get a full score because of the time constraints. The average for the top 33% was 64.60.

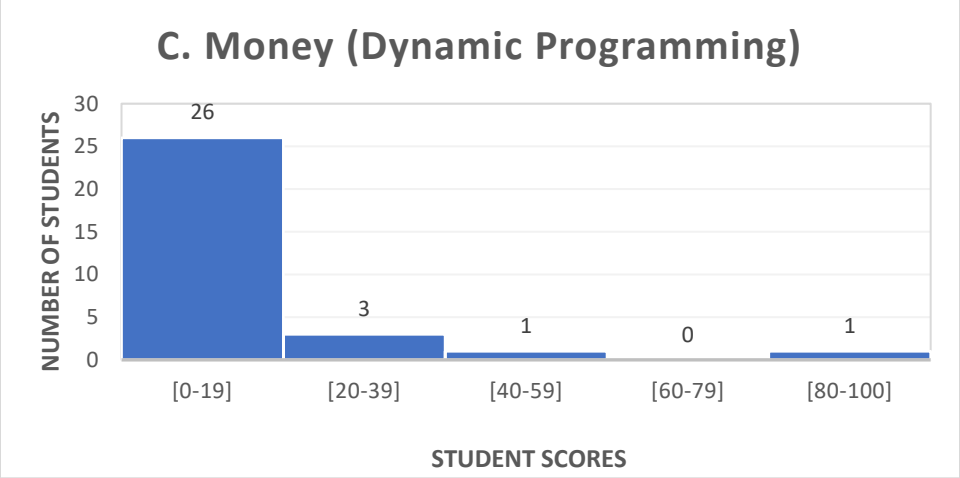


Figure 71: Histogram for task Money

Abbreviated task statement: Given N banknotes help Yuki and Nazima split the money equally. If there is any money left after the initial sharing, they can invest and double the remaining amount and then split it again. How much money will each one make?

I introduced this dynamic programming task to assess the students' knowledge of the identified threshold concept. The basic idea was to use dynamic programming to distribute the banknotes evenly among two persons. Moreover, they needed to distribute the N

[12] A subset of a graph that connects all its vertices without cycles and with the minimum total weight.

banknotes into two equal parts and the remaining sum of the unused banknotes to be the minimum possible.

A brute-force algorithm could receive 47 points ($N \leq 50$, $\text{sum of banknotes} \leq 1000$) by trying out every possible combination of sharing the banknotes. Since there were three possible ways a banknote could have been distributed (for Yugi, for Nazima, for none of the two), the time complexity of the brute-force algorithm was $O(3^n)$.

From the learning set discussions, it appeared that to improve the previous algorithm, students went through each of the banknotes and gave them to either Yugi, Nazima, or nobody. In every step of this algorithm, they needed to know the current banknote and the amount of money Yugi and Nazima have collected so far. We let function $F(K, Y, Z)$ return the largest possible sum of money that Yugi and Nazima can collect. After we have distributed K-1 banknotes, Yugi's amount of money is Y, and Nazima's amount is Z. Then we can write the following function:

$$F(K, Y, Z) = \max[\, F(K+1, Y, N), F(K+1, Y + val[K], Z), F(K+1, Y, Z + val[K]) \,]$$

Where $val[K]$ represents the value of the K^{th} banknote, and function $F(K+1, Y, Z)$ will be 0 if Y is not equal to Z. This dynamic programming algorithm has a time complexity of $O(N \times T^2)$ where T denotes the sum of all of the banknotes.

Some students noticed that it was adequate to keep track of the difference (D) of the amount of money Yugi and Nazima have collected so far. The progression goes from state $F(K, D)$ to the states $F(K+1, D), F(K+1, D + val[K])$ and $F(K, |D - val[K]|)$. These denote not giving the banknote, giving the banknote to the person having more money and giving the banknote to the person having less money, correspondingly. Since there are $O(N \times T)$ states, we can conclude that the algorithm has a time complexity $O(N \times T)$ which is satisfactory to receive a full score.

The space complexity of the task was 32 megabytes forcing students to use more efficient data structures for storing the intermediate values.

Student CG managed to get a full score for this task with the following code implementing the previous algorithm:

```
#include<iostream>
#include<vector>
#include<string.h>
#include<stdio.h>
#define ll long
#define rep(i,a,b) for(int i=a; i<b; i++)
#define N 504
#define MAXVAL 200004
#define INF 1e9+7
using namespace std;
int n, val[N],x;
int dp[N][MAXVAL], sum;

int main() {

    ios_base::sync_with_stdio(false);
    scanf("%d",&n);
    rep(i,0,n) {
        scanf("%d",&x);
        val[i] = x;
        sum += val[i];
    }

    rep(j,0,sum*2+1)
        dp[0][j] = INF;
    dp[0][sum] = val[0];
    dp[0][val[0]+sum] = dp[0][sum-val[0]] = 0;
    rep(i,1,n) {
        rep(j,0,sum*2) {
            dp[i][j] = val[i]+dp[i-1][j];

            if(j + val[i] <= sum*2)
                dp[i][j] = min(dp[i][j],dp[i-1][j+val[i]]);
            if(j -val[i] >= 0)
                dp[i][j] = min(dp[i][j],dp[i-1][j-val[i]]);
        }
    }

    ll ans = dp[n-1][sum];
    ans = ans + (sum - ans)/2;
    printf("%d", ans);

 return 0;
}
```

Overall, 21 out of 31 students (67.74%) managed to solve at least one subtask correctly, even if the task's complexity ratio was 54.32, the highest in the contest for an identified threshold concept (dynamic programming).

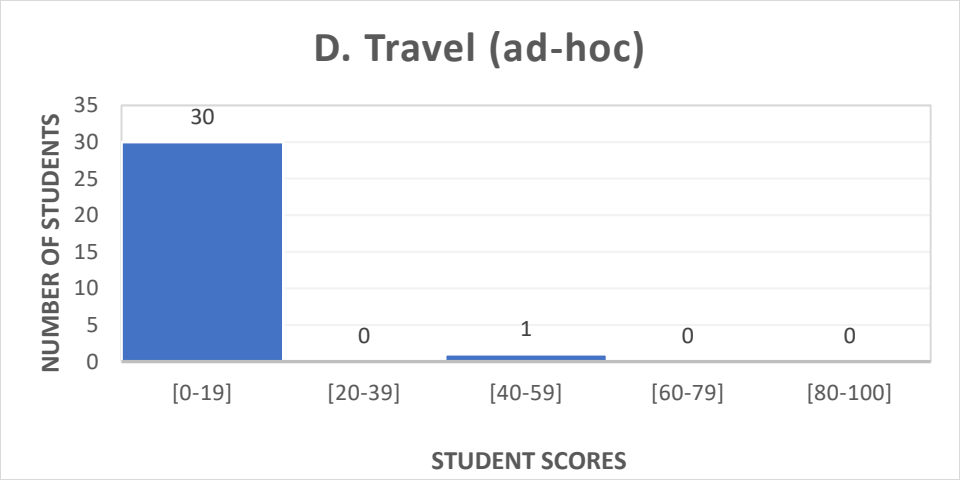


Figure 72: Histogram for task Travel

Abbreviated task statement: Given N cities that are either exciting or boring, connected with N-1 roads, arrange a minimal possible route for Maria so she can visit exactly M exciting cities.

This task was the hardest of the competition, with an average score of 5.10. Moreover, it was the only task in the contest in which none of the contestants earned a full score. This graph task required running a BFS (Breadth-First Search) from every city until M exciting cities were reached. Then, use the DFS algorithm to find the furthest city out of the M exciting cities since the graph was a tree. The learning set session offered meaningful feedback, and five students solved the task on Michanicos.

5.10 Statistical Analysis of COI Round C

Michanicos link: http://81.4.170.42:8980/training/#/tasks/1?tag=COI2019C

Number of contestants: **19**

Tasks	Average Score	Standard Deviation	Group A (Top 33%)	Group B (Middle 34%)	Group C (Bottom 33%)	Complexity ratio
A. Bacteria	13.32	22.65	30.83	7.29	2.83	10.882
B. Ducks	21.05	24.24	42.50	15.71	5.83	7.286
C. Art	26.32	40.58	76.67	5.71	0.00	nan
D. Flow	27.53	15.56	33.83	31.43	16.67	2.030
TOTAL	88.21	80.61	183.83	60.14	25.33	**7.257**

Table 14: Statistical analysis of the third round

The tasks of the third round were the hardest of all rounds. This decisive round formed our delegations for the IOI and BOI competitions (Table 14). The tasks were similar to IOI tasks in terms of overall complexity and subtasks' distribution of points. The overall

complexity ratio was relatively low (7.257), and the highest point average for all tasks was less than 50%. For this round, I also included a task (Bacteria) that was associated with an identified threshold concept (segment trees). Additionally, two subtasks required a dynamic programming technique, so I included and tested two identified threshold concepts within one programming task.

The overall average score for the third round was 88.21, while the corresponding number from the second round was 65.10 (Table 15). To compare all rounds, these are the total averages calculated from the three rounds of competition:

ROUND	Average Score	Group A (Top 33%)	Group B (Middle 34%)	Group C (Bottom 33%)	Complexity ratio
A	215.79	345.57	216.07	81.28	**4.252**
B	65.10	166.80	35.00	5.57	**29.939**
C	88.21	183.83	60.14	25.33	**7.257**

Table 15: Statistical analysis of all rounds

Even though the third-round tasks were the hardest, the students were able to solve more subtasks and get a much better distribution of points for all the group levels.

Tasks analysis of the third round

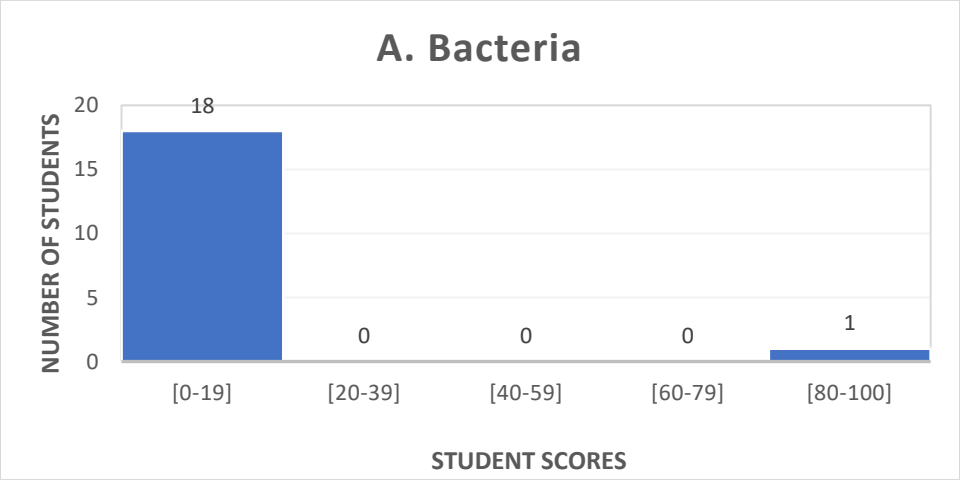


Figure 73: Histogram of task Bacteria

Abbreviated task statement: Help Benjamin maximise the profit of his company which sells bacteria to pharmaceutical research labs within the next N years if he can change the values of X – growth ratio of the bacteria population and Y – selling price of the bacteria on each day.

Task 'Bacteria' was the equivalent of an IOI task. To solve the first two subtasks, the students had to use dynamic programming to calculate the maximum profit after $i-1$ days if they had j bacteria. Then they could get $j * X$ bacteria and $j * Y$ money for that day before moving on to the next day.

The first observation is to consider the profit for two days (i and j) and check which day we have the most profit. The profit for the first day would be $X_1 * X_2 \ldots X_i * Y_i$ and for the other day $X_1 * X_2 \ldots X_{i+1} * Y_{i+1} * \ldots X_j * Y_j$. Then we must check the equation $\max(Y_i, X_{i+1} * \ldots X_j * Y_j)$, which yields the day that gives the most profit so we can sell the bacteria on that day. The time complexity for this solution is O(N).

The second observation is the following: if all of $X_i \geq 2$ then after the first 30 days we will get a value of $X_i * X_{i+1} * X_{i+2} * \ldots * X_{i+29} \geq 2^{30} > 10^9$. So, for all values less than or equal to $N-30$, we can never have an optimal solution because if $Y_{N-30} = 10^9$ and $Y_N = 1$, then $2^{30} * Y_N > Y_{N-30}$. We conclude that it is adequate to check only the last 30 values.

For receiving a full score with the third observation, we can merge consecutive values with $X_i = 1$. After merging, we get the maximum Y_i. It is sufficient to check the last 60 values because it is impossible to have more than 30 merged 1s within the last 30 values, where $X_i \geq 2$. We can use segment trees as the data structure, and we can add information about the current merged values. We can then run RMQ[13] (Range Minimum/Maximum Queries) within the structure to find the answer. So, for a total of Q queries, the time complexity of this algorithm would be O(Q * log(N) * log(10^9)) which would receive a full score.

In general, the complexity ratio of the task was ideal (10.882), and the point average of the top 33% of students was 30.83, which was a lot more than the expectations of the learning set. Overall, one student received a full score and 12 out of 19 students (63.2%) solved at least one subtask correctly.

[13] A technique for finding the minimum/maximum value in a data structure of comparable elements.

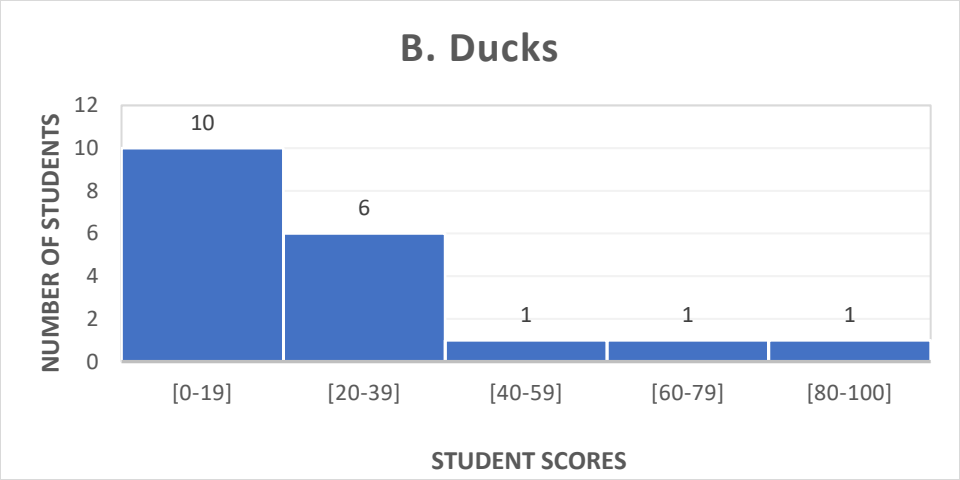


Figure 74: Histogram of task Ducks

Abbreviated task statement:

Benjamin is watching the ducks swimming in circles in the pond. This task has several requests:

- Find if the given sequence of N ($N<10^5$) ducks is valid. Since the ducks go in circles, the sequences (2, 3, 4, 5, 1) and (4, 5, 1, 2, 3) are valid, but the sequence (4, 3, 2, 5, 1) is not.
- Create a replacement sequence to produce the initial sequence of ducks. If we have the valid duck sequence (1, 2, 3, 4, 5) and duck 1 gets tired, we can replace duck 1 with duck 6. Then a valid sequence of the ducks would be (6, 2, 3, 4, 5).
- Count the number of replacements in the replacement sequence that produces the initial sequence.

Task 'Ducks' could have been solved using three different implementations tackling each subtask. First, the students had to decide what a valid duck sequence was. For all the subtasks of this task, there can only be two options. If we observe any of the numbers from 1 to N, then the sequence is fixed, and we can use it to find the original duck for all of the positions. Otherwise, we have the following two constraints: every duck that appears must be in the correct position, and two ducks cannot have the same number.

To consider how to create a replacement sequence and count the number of replacements if the sequence is not fixed, we can choose it arbitrarily. Then the new ducks available will be from N+1 up to the largest possible number. So, to find a valid replacement sequence,

we must decide which new duck must replace each tired duck. Each new duck larger than N can be assigned to a position where the number of the final duck is bigger, which can determine the order of replacements. For the last subtask with too many new ducks to be replaced in the sequence, we can handle these in groups using fast exponentiation[14].

Overall, task 'Ducks' produced a great distribution of points (top 33% - 42.50, middle 34% - 15.71 and bottom 33% - 5.83). In total, 15 out of 19 students solved at least one subtask (78.95%), which was the highest percentage in the contest even though other tasks had a higher point average.

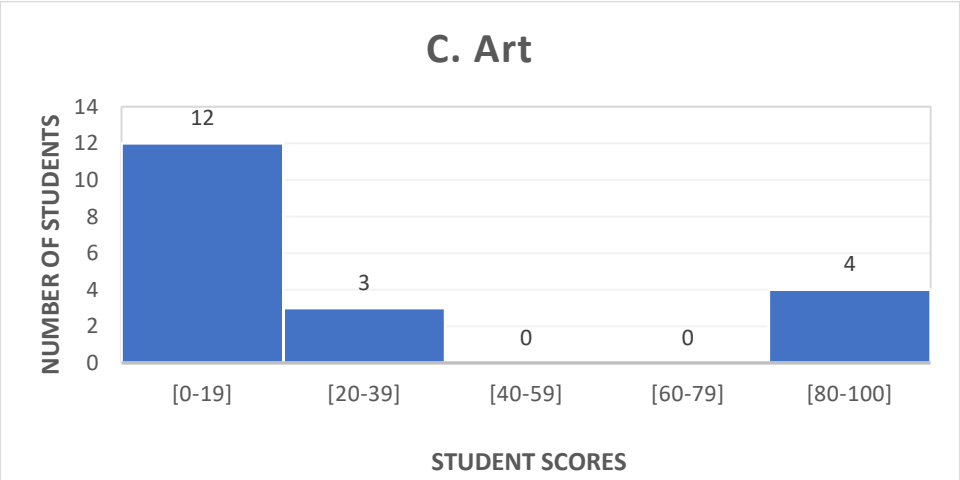


Figure 75: Histogram of task Art

Abbreviated task statement: Given two images of circles with N ($1 \leq N \leq 10^5$) radiuses drawn in each circle, determine if the two images are the same. For example, in Figure 76, if we rotate the image on the left by 45 degrees clockwise, we will get the image on the right.

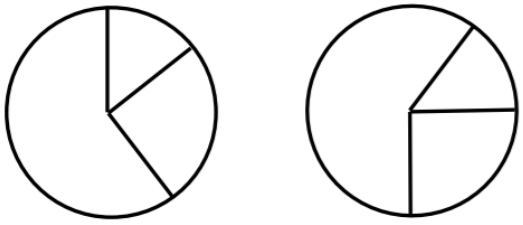


Figure 76: Task Art circle images

For solving this task, students had to check if the two sets of radiuses from the images could be matched with a rotation of either image. Since the radiuses were represented as

[14] A technique for fast calculation of large integer powers of a number.

numbers ($R \leq 360{,}000$), we could sort the radiuses in increasing order and find the differences. To determine if we could get the same image by rotating one of the two images, we could check if sequence S1 was a substring of S2. The KMP (Knuth-Morris-Platt) algorithm can perform this operation in the time complexity of O(N). Task 'Art' created a statistical paradox. Only 8 out of 19 students (42.10%) solved at least one subtask. Four students who knew how to implement the KMP algorithm earned a full score. Eleven students got zero points as they could not implement a brute-force solution to get the first subtask (N<100). Therefore, the point averages for this task were: top 33% - 76.67, middle 34% - 5.71 and bottom 33% - 0.00.

Even though only eight students received points on this task, the point average ranked second in the competition with 26.32. The standard deviation was the highest in the contest, with 40.58. Interestingly, when I analysed the students' code with the action learning set, there were four submissions in which the students did not receive points simply because they failed to check the global rotation correctly. Student CK was positive that his brute-force solution should have received at least 10 points for the first subtask. A closer look at his submission in the analysis session revealed that he failed to check on the radiuses' rotations correctly. He later managed to solve it correctly on Michanicos.

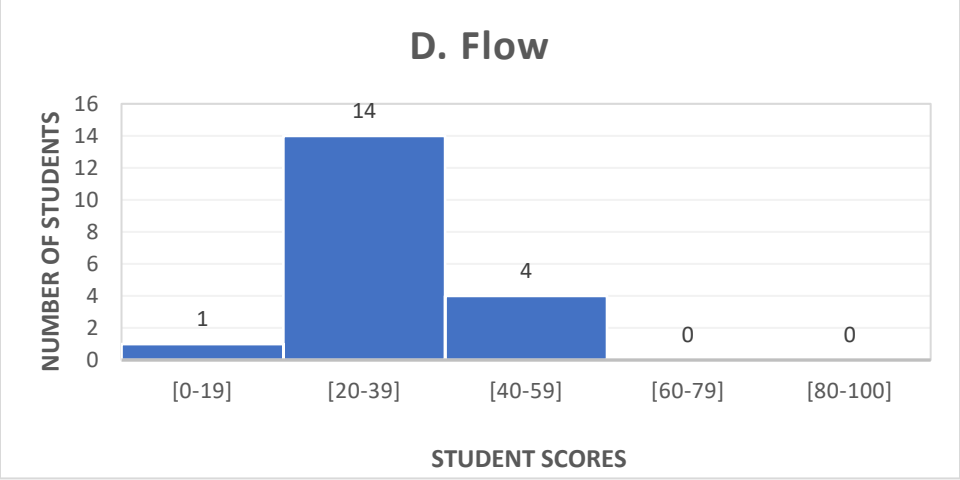


Figure 77: Histogram of task Flow

Abbreviated task statement: This is an output-only task. You need to simulate the game of 'Flow' (Figure 78). In this game, you have to connect the two dots of the same colour with a line without overlapping with any other lines of a different colour. You should not submit any

code with this task. Only submit the output files that correspond to the given input files. You are given three integers with each input file: the dimensions of the grid (N x M) and the number of dots present (K).

Figure 78: The game Flow

The action learning set suggested including an output-only task as there was a task of this type in previous IOI competitions. I have also included a lecture on output-only tasks in the Easter camp, along with the additional lectures on the threshold concepts. The scoring format was as follows: each output file could get up to 10 points with the scoring format of $\text{ceil}\,(10 * (S/K)^2)$, where S was the number of correctly connected dots.

To better illustrate what the students had to do, this is the input file 1:

```
6 6 4
0 0 0 0 2 4
0 0 0 0 0 0
2 0 0 0 0 0
4 0 0 0 3 1
0 0 0 0 0 0
3 0 0 1 0 0
```

Due to its small size, it could have been solved by hand to give an easy 10 points:

```
0 0 0 0 2 4
0 0 2 2 2 4
2 2 2 4 4 4
4 4 4 4 3 1
0 0 3 3 3 1
3 3 3 1 1 1
```

The task 'Flow' awarded the most points in total in the contest (523), which was 31.2% of the total points awarded. The point average of the bottom 33% of students (16.67)

was the second-highest from all tasks, not only in the third round but in all rounds. It ranked second only to the preliminary task. Overall, student performance increased in local competitions due to the COI framework's integration into the training course. Performance data from international competitions are presented next.

5.11 International contest participation

In August 2019, we travelled to Baku, Azerbaijan, to compete in the IOI 2019. Unfortunately, we had to travel with only three contestants instead of four due to the illness of one of the students. These are the results of the Cypriot delegation:

Rank	Username	Total	Rect	Shoes	Split	Line	Vision	Walk
185	CYP1	229.91	37	100	18	19.91	55	0
213	CYP2	204.6	25	85	40	2.6	52	0
230	CYP3	186	8	100	11	12	55	0
	TOTALS	**620.51**	70	285	69	34.51	162	0

Table 16: Cyprus results from IOI 2019

The following table shows the results of the Cypriot delegation in IOI 2018 in Japan:

Rank	Username	Total	Combo	Seats	Werewolf	Doll	Highway	Meetings
176	CYP1	176	100	0	34	16	5	21
223	CYP2	132	100	11	15	6	0	0
284	CYP3	52	30	0	0	6	12	4
313	CYP4	15	10	5	0	0	0	0
	TOTALS	**375**	240	16	49	28	17	25

Table 17: Cyprus results from IOI 2018

Unfortunately, in both of these contests, Cyprus failed to win a medal, but the comparisons of the performance data from both competitions provided very significant findings:

- IOI 2019: Gold medals: 28 (≥ 414.75 points), Silver medals: 54 (≥ 329.18 points), Bronze medals: 81 (≥ 250.19 points).
- IOI 2018: Gold medals: 29 (≥ 336 points), Silver medals: 55 (≥ 272 points), Bronze medals: 83 (≥ 187 points).
- The total points for Cyprus from IOI 2019 had a 65.47% increase from the previous year, even with one less contestant.
- The user CYP4 from IOI 2018 was user CYP1 in IOI 2019. He produced a remarkable total point increase percentage of 1432.73%.

- Our highest-ranked contestant in IOI 2019, although he scored 53 points more than our highest-ranked contestant in IOI 2018, ranked 185th out of 327 contestants while the other ranked 176th out of 335 contestants.

Arguably, the most accurate depiction of threshold concept acquisition by the COI students was when our team encountered the task 'Arranging Shoes' on the first day of IOI 2019. The task was optimally solved by using segment trees, one of the identified threshold concepts of the study. This is the abbreviated task statement:

Arranging shoes

Adnan wants to rearrange N shoes into a valid arrangement. For this purpose, Adnan can make a series of swaps. In each swap, he selects two adjacent shoes at that moment and exchanges them. Determine the minimum number of swaps that Adnan needs to perform to get a valid arrangement of the shoes.

For obtaining a valid shoe arrangement, the greedy approach is optimal since the pair of shoes handled in this way will not interfere with any further swaps. The process can be repeated until the arrangement of the shoes becomes valid. This can be modelled naively in quadratic time, which solves most of the subtasks, or more efficiently in O(NlogN) using a segment tree. Segment trees have been given particular attention since I have identified them as a threshold concept of competitive programming. The Cypriot delegation scored 285 out of 300 maximum points available. The point average for our team on 'Arranging Shoes' was 95, while the IOI average on the same task was 74.47. In the following tables, the results of the Cypriot delegations are compared to different groups of IOI contestants. Specifically, students who won a gold, silver, or bronze medal and non-medallists.

Group	Average
Gold medallists	98.93
Silver medallists	98.52
Bronze medallists	90.74
Non-medallists	54.82
All	74.47

Table 18: Task 'Shoes' results (IOI)

Username	Totals	Shoes
CYP3	229.91	100
CYP2	204.60	85
CYP1	186	100
CYP total	620.51	285
CYP average	206.84	**95.00**
IOI average	245.20	74.47

Table 19: Task 'Shoes' results (CYP)

In August 2019, Cyprus participated in EJOI/JBOI in Maribor, Slovenia. We travelled to Slovenia with five contestants, all under 15. The reason for taking an extra contestant was to better prepare our teams for 2020. Moreover, to allow one of our most promising contestants (Student CYP5 is only thirteen years old and already has the most submissions on Michanicos) to participate in an international programming competition as early as possible. We won four bronze medals in the JBOI competition but what was very reassuring for the future of the COI learning community was that student CYP5 scored the most points.

Rank	Username	Total	Xoranges	Rank	Covering	Tower	Colouring	Adventure
55	CYP5 (unof)	107.1	12	40	5	6.1	10	34
56	CYP1	103	30	40	5	6	0	22
58	CYP2	96	12	40	0	10	0	34
61	CYP3	84	12	40	0	10	0	22
65	CYP4	68	0	40	0	6	0	22
	TOTAL	458.1	66	200	10	38.1	10	134

Table 20: Cyprus results from EJOI/JBOI 2019

In September 2019, a Cypriot delegation travelled to Athens, Greece, for BOI 2019. I had the honour of being selected for the scientific committee of the competition, and the task I proposed (Icarus - Appendix 7) was used on the first day of the competition. Even though two of our students were participating for the first time, we managed to win a bronze medal (CYP1) in this competition as well. Here are the full results of our delegation:

Rank	Username	Total	Icarus	Fishermen	Dictionary	Memory	Roadtrip	Tennis
27	CYP1	160	59	11	26	46	7	11
35	CYP2	69	7	11	26	18	7	0
39	CYP3	44	0	11	8	18	7	0
41	CYP4	39	0	0	8	7	0	24
	TOTAL	312	66	33	68	89	21	35

Table 21: Cyprus results from BOI 2019

The final action learning set session verified that the COI framework's integration into the programming course directly prompted the improved students' results in international competitions (Chapter 6).

5.12 Training systems from IOI participating countries

The goal of the discussions with IOI colleagues was to share ideas on how each country trains its contestants and focused on three aspects:

- What content and resources are available for IOI preparation, and which ones do countries use/recommend?
- What process is the most effective for selecting and training students?
- What other tools are available?

I compared the findings with the COI framework standards to reflect on positive and negative features. The findings (Appendix 13) can be synopsised in the following:

- Public-schools' Computer Science curricula are not appropriate and do not emphasise programming. Clearly, this is a global issue, and it requires much more attention. In most countries, private schools are responsible for IOI participation.
- IOI past contestants (alumni) have been a crucial component in the training systems of most countries. Their duties involve task setting, preparing lectures and leading the delegations.
- Localised online judges have been implemented where possible. Wherever this is not possible, the use of platforms such as Codeforces, Spoj and Usaco, is highly recommended.
- The earlier the students begin to learn competitive programming, the better. If students can be recruited at an early age and assigned a more accessible and easier task set, they will have more experience and, therefore, better chances for success at IOI.

5.13 Reaching the post-liminal space

I interviewed the IOI 2019 delegation (focus group) after their participation in Azerbaijan to investigate the post-liminal space. These four students, including the one who did not travel due to sickness, had succeeded in the local competitions and were selected to represent Cyprus at the highest level. I wanted to determine the connection with the pre-liminal space and how liminality was negotiated. The data from the semi-structured interviews (Appendix 4) was supportive in understanding how the students, particularly the two that have participated in IOI for the last time, have made the transformation and

acquired the qualities reported in the literature. Students who reach the post-liminal space are expected to be accustomed to an alternative way of thinking. This transformation is lasting since it is not just cognitive but also epistemological and ontological (Meyer and Land, 2005; Mostrom et al., 2009). Reaching the post-liminal space implies they have acquired new methods of knowing and can think like computer scientists and accomplished competitive programmers.

The responses from the IOI delegation provided significant indications for their successful transformation to complement their performances. The most natural feature to identify was a noticeable ontological shift, evident in their ability to explain the threshold concepts they once struggled with to their youngest and more inexperienced peers. The methods they suggested using to teach the concepts to students who were stuck in their learning process were the same ones they have used to navigate liminality effectively. Their determination and work ethic proved the connection with the pre-liminal space, reinforcing the importance of active engagement in the learning process. What stood out was their predisposition that whatever was not clear initially would eventually become clear after time. Based on the feedback of these students, the relationship between the pre-liminal variation and the implications of the proposed framework structure and its methods of engagement is undoubtedly affecting the students' cognitive development.

Students' and teachers' perceptions for mastering the threshold concepts tend to agree when students reach the post-liminal space, as I verified with the action learning set. The IOI 2019 team reported that for students to reach a transformative state, they must demonstrate not only a quantitative improvement in terms of points accumulated but also a qualitative trend to solve IOI tasks with recurrently optimal approaches. The emphasis, once again, is on the significance of the quality of programming tasks. The students confirmed that when a specific set of tasks is used as a form of engagement and an appropriate method of assessment is utilised for understanding the conceptual struggles students face, we can verify the effects of the successful transformation.

Furthermore, the IOI delegation made suggestions on how to improve the framework's effectiveness with suggestions for efficient threshold concepts' training, future

programming tasks, lecture topics selection and competition rounds format. They have volunteered as task setters and deputy leaders in future IOIs and expressed a genuine desire to give back to the community. No longer as students but as expert collaborators, ensuring that the community that has allowed them to reach an otherwise unreachable destination will continue to do so for future generations of competitive programmers. This is a solid confirmation of the students' transformation as the COI framework challenges them to discover their new identities within their learning community.

5.14 Issues under consideration and research limitations

Several issues surfaced with the framework's implementation in the teaching practice, and they are presented here:

1. As this was a project combining action research and mixed methods within an educational programme, there is an absence of control vs experimental group data. My main objective was to prepare all COI students for IOI participation while completing this project. If I used a control group, it would have been unethical, and I would have been under scrutiny for not giving my students equal opportunities for successfully competing in the competition rounds. A proposed scenario where a control group was selected and asked to participate without using the Michanicos code-evaluation platform was quickly rejected. This scenario would presumably not allow several students to pass the first round of the competition, and it would have given a definite advantage to the experimental group.

The nature of the produced data was such that no control group was required to provide the corresponding data. In effect, I had the data from previous IOI participation to compare. This data was a very accurate and valid comparison as the IOI tasks are the most advanced measurement of progress. Therefore, the absence of a control group has not affected the study's validity on what was intended to be measured and analysed and enabled me to provide my students with an equal opportunity to make the delegations.

2. The framework requires frequent testing and feedback provision from both students and teachers to provide additional data sources in the future for considering improvements. It is imperative to continue critically evaluating all aspects of the framework even after this study is concluded. The most important attribute that needs to be addressed is the feedback and communication processes. The Slack social workspace that I used for communication was extremely valuable but not the most appropriate as it was separated from the Michanicos platform. The feedback was vital for helping me understand where the students were located in liminality to reflect and intervene accordingly. In that sense, the imminent upgrade of the platform should include a forum component that would allow the coexistence of lecture notes, programming tasks and discussions/editorials on tasks within the same repository.

3. The focus of feature research for competitive programming frameworks should consider several additional modifications. Comparing students' average scores with a relatively narrow 5-point Likert scale on perceived coding efficiency could easily be applied to more extensive data samples. More expanded self-evaluation data and a larger selection group could test more accurately whether very high or low extreme students' scores could be detrimental to their performances and overall progression's overestimation or underestimation. Moreover, additional methods for data correlation should be considered, especially when conducting a regression analysis, and the proper statistical software package should be used to produce accurate results.

4. The framework was designed with the purpose to support teachers to help their students to develop a genuine understanding of competitive programming topics that included identified threshold concepts. The framework requires active student engagement and structured management of the theoretical material. Therefore, the students must demonstrate the ability to explain a concept to others, to approach it with new ways of thinking and to be able to apply it where applicable, such as in new and unknown programming tasks. Correspondingly, the framework can be perceived as a framework of engagement within the course of study. The students are encouraged not to focus on the memorisation of known patterns but challenged to demonstrate satisfactory progression through new achievements. The methods of engagement are

incorporated within the platform, the associated programming tasks and course material. Future inquiries are needed to ensure that these methods continue to produce epistemological and ontological transformations for the next generations, depending on the educational context.

5. The extent to which the research's findings can be generalised can be confirmed when the framework is used in a similar setting for a different country, preferably one that has never won a medal in IOI. According to Maxwell (2005), the findings of qualitative research regularly face generalisability as there is no justification to support that they cannot be applied more generally. Additionally, the findings can lead to similar theories transferred to other disciplines (Maxwell, 2005).

5.15 Chapter summary

In this chapter, the findings of this research study are presented. Based on the findings, the COI framework has been found to have a considerable and multi-level impact on students' learning. In reply to the research objectives, the way programming tasks on the Michanicos platform engage students with their projected learning objectives, whether they were identified as a threshold concept or not, is of critical importance. The real educational value of the proposed framework can be specified by its effects on the teaching/learning processes and learning acquisition within a single year. It would be critical to examine the long-term effects as well. The research findings are only the beginning towards a greater understanding of the framework's impact on competitive programming education in Cyprus and internationally.

CHAPTER 6: Conclusions

6.1 Introduction

In this chapter, I present the main conclusions of the research and define how the study's findings correspond to the research objectives. Each research objective has been addressed with references to the research findings and my reflections. This research study makes distinctive contributions to Computer Science Education research and particularly to competitive programming education by generating a theory about learning and practice. These contributions include the threshold concepts of competitive programming, a methodology for identifying threshold concepts, and the COI framework built on the framework of Meyer and Land (2003), using elements of framework design by Dabbagh (2005). The empirical data support that the COI framework has the potential to inform theory and practice in competitive programming education and introduces a method that can produce consistent results in IOI. By evaluating student performance in liminality on programming tasks embedded with the identified threshold concepts, the Michanicos code-evaluation platform offers methods of inquiry for assessing submissions and provides accurate depictions of students' strategies.

As the lead researcher, with the support of a community of practitioners, including my colleagues from other districts, the COI alumni and current students, I have designed and evaluated the framework and provided a theoretical and practical outline. The COI framework can be perceived as a system of training that contains several crucial components. It aims to empower students to comprehend threshold and other complex programming concepts and support their teachers to communicate them more effectively. As part of an extended action learning set, the COI alumni were used as an expert group to identify the threshold concepts and support the refinement and adjustment of the framework's components. Based on the empirical data, the framework improved students' learning in transformative ways (Section 5.13), qualitatively increased their coding efficiency and programming strategies (Sections 5.4-5.5) and successfully prepared them for IOI participation (Section 5.11). I have shared the COI framework with the IOI community with my presentation at the IOI conference in August 2019 in Baku, Azerbaijan (Eracleous et al., 2019).

6.2 Main conclusions

The purpose of the study is to empower the next generation of Computer Science experts by helping students comprehend complex programming concepts. The research aims to enhance the pedagogy and the teaching practice for the Cyprus Olympiad in Informatics by developing and evaluating a practical framework for competitive programming education and introducing the Michanicos platform under the authority of the Ministry of Education of Cyprus. The proposed framework intends to redefine the teaching and learning processes within the context of its discipline and to become a distinctive contribution to knowledge. The four research objectives provided the outline for aligning all of the framework's components and supported their validity. I will analyse the conclusions reached with each research objective separately.

RO 1: Investigate and identify threshold concepts in competitive programming and how they relate to the pre-liminal and liminal variations of students' learning.

I discovered the theoretical and pedagogical value of the theory of threshold concepts in classifying essential knowledge within my discipline and how it supported me in managing what was essential. Threshold concepts represent the essence of competitive programming as each one holds its own system of ideas or ways of thinking that enable students to acquire knowledge. In other words, threshold concepts represent a way to define critical moments in the learning trajectory, moving a student forward into a new area of understanding.

Using the threshold concept theory in the empirical study made it possible to identify conceptual knowledge that represented learning portals. It also created a depth of meaning and clarified potential implications for how I teach the concepts of dynamic programming, segment trees and recursion. Because one of the research's research objectives was to investigate the process of integrating a framework by the Cyprus Olympiad in Informatics, these implications not only created new knowledge for competitive programming but also developed a solid theoretical foundation on which to base further studies. For example, the empirical study elicited evidence of threshold

concepts negotiations to develop knowledge, mental models and strategies by exploring the learning experiences of highly efficient competitive programmers.

The conceptual difficulties of programming students indicate troublesome knowledge, and the possibility of transforming the students' perspective holds excellent potential for teachers. My task was to identify the source of these epistemological barriers and subsequently free up the blocked spaces. This was accomplished by redesigning the curriculum and programming tasks through scaffolding, providing worked examples and support materials, and peer collaboration.

The Threshold Concepts model (Meyer and Land, 2003) is a valid model for competitive programming education and deserves much more attention across disciplines because it can turn our attention on the topics that will probably obstruct our students' learning. Moreover, the Threshold Concepts model makes inquiries such as 'What do you find hard to understand?' rather than 'Am I a good teacher?'. Therefore, teachers should carefully consider the feedback and insights of their students as learners. Meyer and Land synopsise the most significant impact of the model:

> '...the theoretical significance of this proposed conceptual framework lies in its explanatory potential to locate troublesome aspects of disciplinary knowledge within transitions across conceptual thresholds and hence to assist teachers in identifying appropriate ways of modifying or redesigning curricula to enable their students to negotiate such epistemological transitions and ontological transformations, in a more satisfying fashion for all concerned' (2005, p. 386).

Furthermore, the threshold concepts cannot be taught using traditional ways. We must develop methods of inquiry that will allow us to investigate the variation in students' negotiations with the threshold concepts in somewhat distinct ways (Meyer and Land, 2005). Students should find themselves in situations where they must demonstrate a new way of thinking to solve a complex problem. Knowing the notions that constitute a particular threshold concept can provide the teacher with a framework to be taught

effectively. Our goal must be to accurately measure the students' level of understanding of these concepts, specifically during their negotiation with liminality.

The Threshold Concepts model has enabled the COI educators to look at the threshold concepts through the students' eyes and navigate liminality with each of them, not in the typical teacher/student relationship but in a more cooperative manner using action research. During the study, the discussions and interventions were performed daily and even though, for some students, it was a prolonged and troublesome period, in a way, it empowered them for later success. Not all students have managed to cross liminality within the one year of the study. Some students failed to qualify, and they reported that they had not reached a level of understanding where they felt comfortable with the threshold concepts.

Students who failed to create a dynamic perception of their source code and the notional machine representing its essence got stuck in the liminal space, powerless to advance and develop new ways of thinking about the concepts. For these students, an analysis of the statistics from the platform reported 37% fewer solved tasks on average compared to the students who demonstrated a solid understanding of the concepts (Section 5.4). It will be interesting to monitor the students' scores and attitudes in future years of participation. The maximum troublesome period of liminality reported in the study might be extended to new levels.

The pre-liminal and liminal variations are two of the most crucial notions of the Threshold Concepts model. They are critical in answering the question: Why do some students negotiate liminality successfully, and others do not?. Meyer and Land (2005) consider the pre-liminal variation as potentially essential for improving our understanding of this matter. Both epistemological and/or ontological influences may affect the pre-liminal space. Land (2005) suggested that to improve our ability to understand threshold concepts from the perspectives of teachers and students, we must develop procedures of inquiry that will empower us to examine the variations in the negotiations of students in particular and distinctive ways. As I analysed throughout this project, these variations exist in all three states (pre-liminal, liminal and post-liminal).

A critical aspect of the pre-liminal variation, as evident from the empirical data, was the willingness of the students to endure a troublesome period of learning and have the opportunity to represent their country in international programming competitions (Section 5.2). Additionally, the IOI delegation reported traits such as willingness to succeed, attitudes on learning, determination and perseverance, which appear to affect the negotiations with liminality (Section 5.13). Therefore, even though the results are not convincingly overwhelming, these traits indicate what needs to be investigated within the pre-liminal variation to make predictions.

Based on the study's empirical data, there were strong indications for the liminal variation, and the Michanicos platform provided the means for measuring (Section 5.4). Performance data and scores from competitions, the number of tasks that students have solved associated with threshold concepts and the corresponding time frame and slack interactions constituted relatively dependable indications of the liminal variation for a possible successful negotiation of the concepts (Section 5.4). Three out of four students who solved more than the average number of solved tasks made the IOI 2019 delegation (Section 5.13). The remaining one solved less than half of the average mark but still made the team. However, the qualitative analysis of the student's submitted source code indicated an elaborated optimal code on most occasions (Section 5.5).

In collaboration with the COI community, I used this information to adjust the curriculum to emphasise the threshold concepts and create the associated programming tasks through the action learning set. However, I kept in mind that not all IOI tasks in future competitions will include the threshold concepts identified in this study. In essence, special attention was given to the identification and the negotiation of the threshold concepts, but, at the same time, the other topics not identified as threshold concepts were not neglected. In conclusion, my goal was to identify the troublesome aspects of disciplinary knowledge and, through the provision of material, technology, support and collaboration, empower successful negotiations through liminality, stipulate the ontological shift in students and the perspective shift in teachers.

To summarise, there are three valuable aspects that the Threshold Concept theory provides for competitive programming. First, theory offers a suitable description of what constitutes a successful competitive programming student. Second, the epistemological concepts of pre-liminal and liminal spaces can support research on the qualities of a successful student. Last, the research is done within a learning community, giving deference to the teacher's disciplinary knowledge where any issues are continuously evaluated and resolved within specific disciplinary settings (Meyer and Land, 2007).

The theoretical value of the Threshold Concepts model can be found within the process of discovering complex topics of disciplinary knowledge, identifying and embedding the specific threshold concepts within the teaching process and, therefore, empowering educators to redesign or modify their curricula and enhance their teaching methods. Educators must inform the students about liminality and threshold concepts, increase their understanding of their current liminal states and, most importantly, enable them to deal with the negative emotions associated with the transformative journey (Meyer and Land, 2003). Students must be capable of defining their current ways of thinking and practising and be driven to change those ways. Students must be actively and consciously engaged when negotiating each threshold concept to master it completely.

Both the pre-liminal and liminal variations are crucial for students' engagement in learning. Educators can be benefitted by identifying the origins of confusion, places where students get stuck and issues with epistemological beliefs. If all of the above can be accommodated within the context and time constraints of course design, they can greatly impact student learning.

RO 2: Investigate the process of integrating a framework by the Cyprus Olympiad in Informatics for preparing students for the International Olympiad in Informatics participation and the practical context of this decision.

I investigated the integration of the COI framework in the programming course responsible for forming the national delegations. The COI framework is based on the transformative interactions between the pedagogical model of a learning community

driven by situated cognition and constructivist views of learning, the instructional strategy of PBL that supports collaboration, and the online learning technologies. Moreover, a fourth component was the worked examples used to initiate the process of scaffolding. I evaluated the framework's impact on student learning using the action research methodology. Since the alumni identified the threshold concepts, special attention was given to assessing the students' negotiations with liminality, particularly with the Michanicos programming tasks embedded with the identified threshold concepts. One of the most critical findings from the threshold concept inquiry was that each student takes a distinct path to learning and possibly struggles in different places (Section 5.4). These findings propose that there should be no standardised order of topics or assignments, but rather the progression should be flexible enough to facilitate individual students' needs.

Action research has been a practical way of meeting the individual needs of my students when they negotiate the liminal space. Whenever I engaged with action research, I moved through two authentic liminal spaces: action and research. When I used the action research cycle, I could identify what the effect of my teaching was by connecting the two states above. The liminal space of research allowed me to observe and reflect by questioning, analysing and synthesising. In contrast, the liminal space of action allowed me to plan and act by reaching conclusions, adjusting, designing, implementing and administering.

Most importantly, action research has enabled me to discover what kind of knowledge (deep or superficial) could be created from this framework. How to acknowledge the significant changes in students' learning trajectories, and what could be known about these changes by anyone other than the students? These were relevant concerns, but an even more relevant question was: What was the impact on learning within this type of framework for my students within this specific context?

With the action research methodology, I adjusted the teaching strategy accordingly throughout the study. The way my students learned was unique, and their learning trajectories within the liminal space varied as they would get stuck at different places. The students' disposition to engage with the threshold concepts and negotiate the related

liminal space equally differed. For advocating action research, yet once more, how was I supposed to make any sense of how knowledge was gained through liminality if I based the notions of knowledge acquisition only on established pedagogical approaches or literature review on the work of others? Therefore, the triptych: theory, empirical data and research findings, which fitted perfectly with the action research methodology, was used to meet the underlying research objectives.

I have used the investigated pedagogical models, the IOI training systems and my previous knowledge and merged them with the relational and experiential knowledge produced by the action research. Concurrently, I was considering how the teaching and learning processes could be improved within the course of study. The quality of the action research outcomes is certified when the study did as much as possible, considering the context and the specific circumstances under which it was conducted. Additionally, when action research was combined with mixed methods, it produced scientifically sound and reliable results based on the provision of quality student/peer feedback from the action learning set and the focus group. The results reported are only based on the previous year. It will be of great interest to further investigate the framework's contributions in the years to come and strengthen the study's validity even more with sustained involvement. After all, cycles in educational action research should never really end but rather continue with new cycles of action each school year to improve my teaching practice even more.

The conclusion that I was able to reach concerning the applicability of the framework and its components was that it needs to be utilised in a competitive environment with clear goals and objectives for different age levels (Section 5.6, 5.12). The framework has been built on a solid theoretical basis (Meyer and Land, 2003; Dabbagh, 2005), and it must not be perceived as a standard teaching approach. Solving multiple programming tasks and participating in numerous programming competitions might be suitable for the university level. The former is an adequate enhancement to the university lectures but not appropriate with younger and novice programmers. With high school students, the goal must be to develop their problem-solving ability by challenging their desire to compete and succeed within a learning community.

The instructional strategy of PBL offered authentic opportunities to the students to foster active learning, provided scaffolding, supported knowledge construction and associated the learning with worked examples and programming tasks. The design of PBL with scaffolding in a collaborative learning environment to develop problem-solving skills was feasible with the use of online technologies. Worked examples as notional machines have supported the mental model acquisition, assisted by a range of tools, visualisations and features of the rich programming environment.

One of my framework's most significant characteristics is the transfer from an exclusively teaching method to a method of learning and progression within a community of learning. This is an irreplaceable effect of a collaborative process for teachers and students as they both support the growth of the potential of others. The framework supports different age groups to reach their estimated upper bound of learning (a programming threshold) using different levels of scaffolding through three phases of the curriculum.

For students to reach a new level in their stages of progression within the framework, they must successfully go through each of the programming phases and meet the associated programming thresholds of their age group. This achievement will allow the systematic development of students, continually expanding their abilities by solving complex tasks and challenging their unique aptitudes to apply their knowledge in practice. Scaffolding was adjusted as the students demonstrated their achievements and, consequently, when they reached the required levels of understanding, their programming threshold became a threshold concept. The progression for each student must be at an individual pace as programming concepts associated with different age groups can be challenging to negotiate, just like threshold concepts. The teachers supported the students with their progression by assigning programming tasks, providing scaffolding, and assessing progress on the Michanicos platform. These programming tasks were considered progressing tasks, and the associated competition rounds represented the programming threshold for each age group.

As a training system, the COI framework can be applied to the learning process as early as possible. The programming threshold for elementary school students (ages 8-12) was

the first-round competition and the associated phase-one material. They received maximum support from the community during their interactions with the course material and programming tasks. The focus was on improving their algorithmic thinking and their ability to use the computer to solve problems and implement new ideas. For gymnasium students (ages 12-15), the programming threshold was the second-round competition and the associated phase-two material where they were initially introduced to an identified threshold concept (dynamic programming). The focus was on their preparation for competing in the European/Junior Olympiad in Informatics. Lastly, for lyceum students (ages 15-18), the programming threshold was the third-round competition and the associated phase-three material. The programming threshold for this age group was the identified threshold concepts, and the focus was on their preparation for the IOI competition.

RO 3: Evaluate the teaching and learning processes of competitive programming using a code-evaluation platform with competition-type programming tasks embedded with identified threshold concepts and measure the effect on students' strategies.

Although there are not many programming courses explicitly for competitive programming, similar to the CS3233 module taught at the University of Singapore (Halim and Halim, 2009), the use of online judges in universities has increased rapidly over the years. An online judge allows the students to submit their computer programmes and automatically assesses their source code without a human presence. The literature revealed that this approach enhanced the contestants' ability for self-assessment (Garcia-Mateos and Fernandez-Aleman, 2009; Dagiene and Skupas, 2011).

The Michanicos platform has been the cornerstone of the framework all along. It has been the most significant component that my community has expected for quite some time. Indeed, there were many other commercial options available that have always been helpful in the programming course. Nevertheless, using a platform designed and implemented within the community was undoubtedly a breakthrough for my teaching procedure and a valuable resource for secondary and tertiary education.

The most critical role of the platform in the teaching and learning processes was the establishment of the COI as a decentralised and competitive learning community. It brought the teachers and students from the five districts together and allowed them to share their knowledge and accomplishments. The platform enabled teachers to create programming tasks, share them with the COI community, assign them to students and analyse the results of the automated evaluation process in a timely and efficient manner.

From a teacher's perspective, this was a critical update from previous methods. I was now able to truly focus on each student's learning trajectory as I monitored students' progress through a single point of reference. The real-time automatic evaluation of tasks has received positive feedback from teachers and students. Teachers were able to utilise a much more efficient assigning and grading process, and students were able to view the results of their programmes with full feedback for each task. The task setting procedure, which used to be a significant burden for my colleagues and me, has become much more effective. Moreover, the feedback from students classified the platform as their personal code repository/portfolio. Students' engagement with the platform has been steadily increasing from the first month of deployment. It has reached a level of weekly interaction for more than 80% of COI students who reported attempting to solve two or more tasks (Section 5.4). By using the platform, as discussed in RO1, I was able to use distinct methods of inquiry to measure the level of understanding of threshold concepts from the evaluation of student engagement and negotiations with the concepts.

Additionally, I have been able to assess the effects on students' programming strategies. The platform has provided quantitative data with the students' achievements and accumulated scores and qualitative data from the easily accessible programme submissions with the associated timestamps (Sections 5.4, 5.5, 5.8-5.10). This feature has been another crucial development as all of the users' submissions are stored within a single repository. Using the repository, I tracked students' progress from their initial submission to their final one and identified the strategies used by both novices and effective student programmers. These strategies revealed the existence of viable mental models and successful negotiations with liminality. Optimal strategies were communicated to novices through the Slack social workplace.

Though it measured only the previous year's progress, the empirical data from the study has been very indicative of the learning environment. Using the learner-centred approach, the students have demonstrated a profound involvement with the course context and the corresponding programming tasks. The students were able to perceive threshold concepts from different perspectives; they were eager to interact/share their knowledge and achievements as well as their struggles/inquiries with their peers/teachers and were adept at producing optimal solutions. Empirically, with the platform interactions, they have improved their programming skills, learning attitudes, and strategies (Section 5.5).

What is equally important is that though most countries have advocated using a localised online-judge system for IOI preparation, less than 10% of the participating countries have developed and implemented one for their training courses (Appendix 13). This percentage proves the complexity of utilising and administering such a system as most countries lack the personnel or expert knowledge for successfully doing so. The Michanicos platform can be integrated into any level programming course. It offers an accessible multi-language interface, can run multiple contests simultaneously and allows multiple administrators to create and assign tasks in multiple languages. Accordingly, using technology in a pedagogical framework that relies on active learning suggests that we need a situated understanding of how students become critical consumers of all the information that technology puts at their disposal. Situated cognition is consistent with the epistemological assumptions of constructivism, which specify that meaning is a function of how students create meaning from their experiences and actions. Deliberate practice is a critical aspect in acquiring programming skills, and deliberate practice generates experience (Scott and Ghinea, 2013).

RO 4: Determine if the degree of related student engagement and motivation with the learning process can improve the learning outcomes within the Cyprus Olympiad in Informatics.

Reynolds (2010) noted that bringing a spirit of play to work and the feeling of exploration and discovery that it instils at the moment improves learning and stimulates improved ways of thinking. Programming students must be motivated so that they will engage

appropriately. Motivation for a programming student includes two key factors: The first is exploring a field that is engaging and stimulating to the individual. The second is doing this in a social environment that encourages social negotiation. The framework incorporates these factors into the COI course by using PBL as the instructional strategy, providing scaffolding and utilising a code-evaluation platform to create a truly collaborative learning environment.

The COI framework became a framework of engagement that promoted collaboration between students to reach the same goal. It has enabled students to reach interpretations of the methods of experts as those think and practice in a community of learning. Active student engagement in the learning process was essential as it enabled students to interpret how computer scientists think and to start to think like computer scientists. The forms of engagement for producing the required transformations in understanding were incorporated within the framework with the associated programming tasks on Michanicos and the collaborations in the Slack social workplace. Carefully designed programming tasks required students to explore complex programming concepts found in university curricula and reach new levels of understanding.

After identifying the threshold concepts, the following step was to make the concepts cognitively challenging for the students. The process included designing programming tasks, embedding the concepts and making these tasks accessible to students through online programming contests. Contests do not immediately guarantee the successful negotiation of concepts or the improvement of the contestants' problem-solving abilities and strategies. The competition's success is highly correlated to the quality of tasks and the value of the feedback the contestants will receive upon contest completion (Combefis and Wautelet, 2014). If the students' engagement involves mere contest participation, the results will not be as expected, and motivation will be affected. Each task must have distinctive characteristics that will retract the student's previous knowledge and generate innovative and optimal strategies to help the student solve it.

The qualitative assessment of the code submitted by the contestants was critical for the learning process. Quantitative assessment in the form of points accumulated cannot be

the only evidence of students' improvement (Ala-Mutka, 2005), specifically for the students stuck in liminality. Forisek (2013) reported that the standardised programming competitions emphasise the application of practical algorithms, and the qualitative feedback presented for a proposed solution is either very limited or missing. To integrate online programming contests in the framework, I used appropriate methods to communicate the qualitative feedback associated with each task after contestants tried to solve it.

The engaging factors of the learning process were critical, and they have been found to affect student motivation. However, this had to be confirmed based on student data and the open-ended feedback on the effects of the platform usage. The feedback from the students for the platform interactions was predominantly positive, but the subjects were competitive programmers in an advanced programming course. Therefore, it will be interesting to investigate the impact of the Michanicos platform with an introductory programming course with novice programming students and a relatively easier set of tasks.

In terms of feedback, students reported having a positive experience within the COI community, and the individual components received high praise from most students and teachers (Section 5.6). To ensure the authenticity of the data, I collected data from students from all districts, most of whom I have never personally met. Therefore, my positioning in the COI community could not have affected the students in any way. Feedback from both teachers and students was helpful to adjust the components for enhancing the learning process and make interventions in the teaching process. The positive experience corresponds to the epistemological stance of the students, as I have determined from their pre-liminal investigation (Section 5.2). Most of the students that participated in the study had a unique level of self-motivation which positively affected their engagement with the learning process and their results. The research showed that student motivation originated from the single and most central outcome of COI participation: learning and representing their country. Their self-drive to be the best student programmers in Cyprus and compete with peers from other countries significantly impacted their negotiations with liminality and the threshold concepts. From the

conclusions of RO1, it was clear that the pre-liminal variation (Section 5.2) was critical in terms of prior subject knowledge and willingness to succeed and navigate through a lengthy learning journey through liminality.

In terms of the learning outcomes, the results of the COI students in local competitions (Sections 5.8-5.10) and the performance of the Cypriot delegations in international programming competitions (Section 5.11) have improved significantly. The 820 total points accumulated by the delegation of Cyprus in IOI 2020 was the highest total we have achieved as a country in thirty years of IOI participation:

Rank	Username	Total	Plants	Supertrees	Tickets	Biscuits	Mushrooms	Stations	Award
97	CYP2	316.94	27	100	11	21	92.62	65.32	*Bronze*
180	CYP4	225.32	19	96	11	9	25	65.32	
236	CYP1	165	5	100	11	0	10	39	
266	CYP3	112.64	5	40	11	0	56.64	0	
	TOTALS	**819.9**	56	336	44	30	184.26	169.64	

Table 22: Cyprus results from IOI 2020

Regarding the learning outcomes in terms of student understanding, most of the literature involving introductory programming courses has reported that most programming students cannot write code in a meaningful way (Carter and Jenkins, 1999; McCracken et al., 2001; Guzdial, 2011). The former issue has not been explained yet (Robins, 2010). Accordingly, teaching students how to write code, assessing learning outcomes and levels of understanding is equally challenging as well. My research was inspired by exploring the usefulness of liminality to meet these challenges. Linking back to the literature review, since the liminal space of a programming concept is the time frame when a student is actively trying to understand the concept but has not yet succeeded, it can be described as a partial understanding of the concept.

Learning to code at the IOI level requires a deep understanding of programming code. At this level, students can demonstrate deep and conceptual understanding rather than superficial understanding (Hattie, 2012). The optimisations of students' code, evident in their strategies, have provided sufficient evidence for a deep and conceptual understanding of the assigned challenging tasks. However, as the instruction aims towards

the students reaching this level of understanding by themselves, it is vital that scaffolding is eventually faded, and the teacher perceives the concepts through the students' eyes.

With the qualitative data gathered from the action research (Sections 5.4-5.5), I have identified three levels of understanding throughout this study aligning with the findings from the literature. In the first level, I have encountered a form of mimicry of the new state. Meyer and Land (2005) identify mimicry as a form of understanding and troubled or limited misunderstanding. Mimicry is a non-intentional attempt to reproduce known information.

The first level is a theoretical (abstract) understanding where code replication is usually the only trait present when students get stuck in the liminal space. In the second level, there is evidence of partial understanding (Section 5.4). Still, the partial ways of understanding usually contain an acquisition of several relations between aspects but not all of them. The second level describes numerous practical ways of understanding, but without a theoretical understanding. Lastly, the third level describes a deep and conceptual level of understanding, where a connection is made between learning about code behaviour and understanding competitive programming. In other words, two aspects of understanding a concept are: knowing its rationale (why you want to know and use the concept) and its application (knowing how to use the concept in unknown tasks). I have verified that meticulously designed programming tasks have supported students to reach this level of understanding. The focus group confirmed the quality of the programming tasks for fostering cognitive development and engagement (Section 5.14).

The teacher's role in the process was crucial as I was the facilitator of students' progression through the levels of understanding by providing different levels of scaffolding with varying methods of task support. By helping the students look into and reflect on their source code, they were able to progress through the levels. Therefore, as a teacher, I often found myself focusing on the outcomes, revealing my inclination to remain with the action instead of the research. However, what was fundamentally important for me as a teacher-researcher was to remain with the action, engage with the theorising, and focus on the research objectives. Consequently, reaching the deep and conceptual level

of understanding was established as a reliable indication of a successful negotiation with liminality and threshold concepts. The latter demonstrated how students' performance qualitatively increased in complexity while mastering associated complicated programming tasks (Sections 5.5, 5.13).

Another important aspect was the measurement of the learning outcomes. The learning outcomes are problematic to measure as there is uncertainty on what constitutes credible means of measuring students' learning (Breslow et al., 2007). Research methodologies used to measure student learning can be subject to bias. Therefore, the best possible practices include the triangulation of the data. Triangulation instructs using a mixture of data sources and inquiring methods to achieve a broader interpretation of the investigated outcomes. Simultaneously, triangulation lowers the probability of systematic bias and chance associations due to a particular method (Maxwell, 2005). In this research, triangulation was an ongoing negotiation between the researcher, the empirical data, and the liminal space described by Meyer and Land (2003).

When there are numerous distinct data sources, it dramatically raises the possibility that the outcomes are precise. The best practices depend on a combination of indirect and direct assessments. The indirect assessments of this study involved feedback from the alumni and educators (Sections 5.3, 5.6), which is not a direct indication of what the students have learned but rather to infer the benefits from COI participation and the training system in general. The direct assessments involved the students' interactions with the Michanicos platform and the assigned programming tasks (Section 5.7).

The accumulated scores for each task set, specifically the ones embedded with the threshold concepts identified, have provided a much better indication of the improved learning and the increase in students' knowledge and skills over time (Section 5.4, 5.8, 5.9, 5.10). Overall, the platform's support and impact on student learning have been measurable and critical. Still, special attention is required for the context and the learning environment intended to be applied within.

To summarise and make the evidence base explicit, the following table (Table 23) presents how the empirical data contributed to the research findings, which group of people provided the data and how they supported the associated research objectives.

RO1
• Students' pre-liminal variation (5.2 – Bebras Students) • Threshold concepts in competitive programming (5.3 - Alumni) • Perceived/Actual coding efficiency with threshold concepts (5.4 – COI Students) • Students' post-liminal variation (5.13 – IOI 2019 delegation)
RO2
• Student feedback on the framework (5.6 – COI students) • Training systems from IOI participating countries (5.12 – Appendix 13, IOI peers) • Issues under consideration and research limitations (5.14)
RO3
• Student feedback on the platform (5.6 – COI students) • Perceived/Actual coding efficiency (5.4 – COI students) • Students' code optimisations and programming strategies (5.5 – COI students) • Performance data from competition rounds (5.8-5.10 – COI students)
RO4
• Student feedback on the framework (5.6 – COI students) • Data on students' engagement (5.7 – COI students) • Performance assessment in international programming competitions (5.11) • Feedback from the IOI 2019 delegation (5.13 – IOI 2019 delegation)

Table 23: Association of empirical data with research findings and research objectives

6.3 Chapter summary

In this chapter, I presented the project's main conclusions specifying how they addressed the research objectives. I evaluated the framework within the COI learning community and the empirical data collected provided valuable insights regarding students' feedback and improved performance in international competitions.

CHAPTER 7: Self Reflection

I completed this research project within the context of my professional role as a CS instructor for the Cyprus Olympiad in Informatics (COI). COI is the organisation responsible for preparing the delegations of Cyprus to participate in the annual IOI, BOI and EJOI/JBOI competitions. As one of the COI teachers since 2011 and the team leader of the Cypriot delegations since 2015, my inspiration for this project was developing, implementing, and evaluating a practical framework to improve competitive programming education.

My studies at Middlesex began in 2002 when I applied for my MProf. About the same time, I started teaching CS in secondary education. As a CS teacher for the past twenty years, I have participated in several research projects in secondary education and the COI community. In 2004, I earned my MProf from Middlesex University with my 'Proposal for the development of a three-year curriculum programme in CS for the secondary schools of Cyprus' (Eracleous, 2004), which initiated the switch of the programming language from Pascal to C++. Like every significant curriculum reform, this shift was delayed substantially, but the ministry finally adopted it in 2016.

To prove the validity of the shift, in the IOI 2019, C++ was used by 97% of the contestants, which validates the importance of using the most appropriate programming language for a competition setting (IOI, 2019). After earning my MProf, my following goal was to pursue a Doctorate in Professional Studies to accomplish my primary academic objective: to teach at the university level. Nonetheless, being part of the COI and the IOI communities with their university equivalent curricula and complex programming tasks gives me a strong belief that I have already reached that level.

Since the IOI topics are extraordinarily challenging and do not exist in any high school CS curricula, the critical question was 'how can I teach these topics to students that are not supposed to learn them?'. These are unapproachable topics based on the age threshold of the students. I had an initial theory on how to achieve this. I have developed the framework and used action research to test it, and I designed the instrument (Michanicos

platform) to communicate it. Based on empirical data, I have successfully managed to improve the quality of the teaching procedure and the students' results at all levels with my teaching practice.

I have always collaborated with peers, alumni, and students within my learning community to improve my teaching practice and student performance, engagement, and motivation. In recent years, an appreciation for programming and its applications has increased steadily. As a technological society, our advancements rely heavily on our capability to educate students in Computer Science. Understanding programming will expand the students' perception of the world and support them in creating new ways of thinking. I have used my knowledge to confront the challenges presented by this study. I did not make this project merely to increase students' performance within a programming course. Instead, the primary goal was to support my colleagues and me in transforming our students and developing their ways of thinking. Moreover, to help them become competent and productive within the academic and technological sectors upon employment and, ultimately, improve the quality of their lives.

My quest for academic and professional development is becoming increasingly important. The ongoing changes in my discipline indicate that what I have studied in the past will probably become unrelated in the following years. Moreover, the volume of knowledge, technological innovations and the quantity of available data continue to grow. In this modern world, I cannot understand everything there is to know, yet access to all of this information is easily accessible online. A great professional has gained more significant and exchangeable knowledge and can apply it much more effectively than a bad one (Moore, 2007). Determining how to communicate the acquired knowledge and make it relevant to others is what I need to accomplish to avoid becoming insignificant.

Presuming that my qualifications will last a lifetime will not progress me on the path of academic and professional development and will make me redundant. Innovative and improved online judges are launched every year, new programming languages are built, and more effective algorithms are created. As an IOI instructor, I must try to keep up with these advances. If I fail to do so, then my competitive learning environment will ultimately

fail to provide the desired level of teaching, causing a decrease in students' performance and motivational levels. Thus, I must learn from my practice, which involves a particular set of competencies. Competencies that I need to preserve and be able to transfer to my students for their future development. Correspondingly, the proposed framework has been in progress ever since I earned my MProf. My work for both my MProf and DProf programmes is tightly connected because the curricula reforms were essential before designing the framework's layout.

My research study has allowed me to reflect on the knowledge and abilities obtained throughout my career. My scientific background has provided me with the knowledge and expertise to design and evaluate a framework to increase students' learning.

My skills have been used throughout the study and can be synopsises as follows:

- I am continuously striving for individual and academic development and keeping informed with innovative scientific developments in education.
- I have achieved profound knowledge of programming concepts in the university curriculum, well above my current academic status.
- I am continually evaluating and adjusting my practices to meet my students' needs and provide them with the knowledge and future they deserve.
- I can design and test educational projects and software.
- I am competent to plan and conduct advanced research projects within my learning communities.
- I am knowledgeable about establishing and following an ethical code of conduct throughout my research to protect the integrity of participants.

As a computer scientist and an educator, I decided to use the pragmatic paradigm (Creswell and Clark, 2011; Morgan, 2014). I believe that no paradigm suggests or forbids either methodological approach. So, since I needed to collect numeric data (student scores) and other forms of data such as source code, feedback, recommendations and views, I have decided to use qualitative and quantitative approaches in supporting ways. My investigation on research reports verified that CS researchers favoured the pragmatic paradigm for assessing programming competence and students' level of understanding.

Furthermore, pragmatism was used by researchers interested in the development of teaching and learning processes and the appropriate use of programming tools in the teaching process (Sheard et al., 2009).

From a constructivist viewpoint, the real issue of programming education is not merely to understand the syntax and the semantics of programming languages. Instead, I want my students to view programming as an ongoing quest that involves the skills of problem-solving and optimal programming strategies. The quest defines an authentic programming genius capable of solving complex and unfamiliar tasks in educational settings. Such settings include the IOI, where students encounter an unknown set of complex tasks every year. Furthermore, the constructivist paradigm offers a solid foundation to build upon as it is continuously supported in Computing Education Research (CER). My priority was to help my students gain a significant amount of knowledge to participate at the highest level of high school programming education and compete successfully against the world's best student programmers. Optimistically, I hoped my students could win medals and receive scholarships from top universities. However, if I had rejected the active role my students played in constructing their knowledge to debate the applicability of a different epistemological paradigm for research purposes, it would have been a treacherous path that I had no intention of taking.

This project study has enabled me to review my knowledge of competitive programming education. Through the research, I have identified threshold concepts in competitive programming that I needed to consider for the COI advanced programming course. I have identified several issues to consider before applying the Threshold Concepts model in my discipline. To use this model, teachers must acknowledge the limitations and the possibilities that the students have. Throughout my academic career, teaching students of all ages has allowed me to recognise the struggles they face, both as novices and as experienced programmers. I can identify particular reappearances of related problems as I have encountered the majority of these through my practice. My work experiences and DProf studies have drastically improved my critical thinking skills and expertise in gathering and investigating data. Manually assessing the performance of source code and

instructing students has enhanced my ability to support them in producing optimal code and navigating through the stuck points in the liminal space.

I am positive that I have contributed to the discipline of competitive programming by introducing the COI framework (Figure 79) and evaluating its impact on learning and the performance of students within my learning community. During this study, I have investigated the training systems and methods of other participating countries, and I have gained valuable knowledge on how the COI framework corresponds to the positive and even negative aspects of existing training modules. I have used the feedback from my fellow team leaders to adjust several components of the framework during the study, and I will continue to do so in future years.

To improve the effectiveness of the COI framework, I have researched the IOI tasks from the previous twenty years to understand the setters' tendencies and cultivate a clear blueprint for task setting. I have authored four programming books (Appendix 9) and multiple programming tasks for international programming competitions such as the Balkan Olympiad in Informatics (Appendices 8 and 12) and the IEEExtreme (IEEExtreme, 2019). I have written thousands of training tasks used for practice at every level of programming education that I have shared on Michanicos, HackerRank and other platforms so that my peers and students can benefit from assigning and solving, respectively. The art of creating appropriate programming tasks to challenge some of the worlds' best student programmers is a whole new independent study on its own.

With the Michanicos platform, I have contributed to the academic and scientific fields by designing the localised online judge. Apart from the COI community, Michanicos can be utilised in secondary and tertiary education. The platform can be integrated into the course of study of any programming course and support teachers and students by providing full feedback on submissions and promoting powerful ways of thinking.

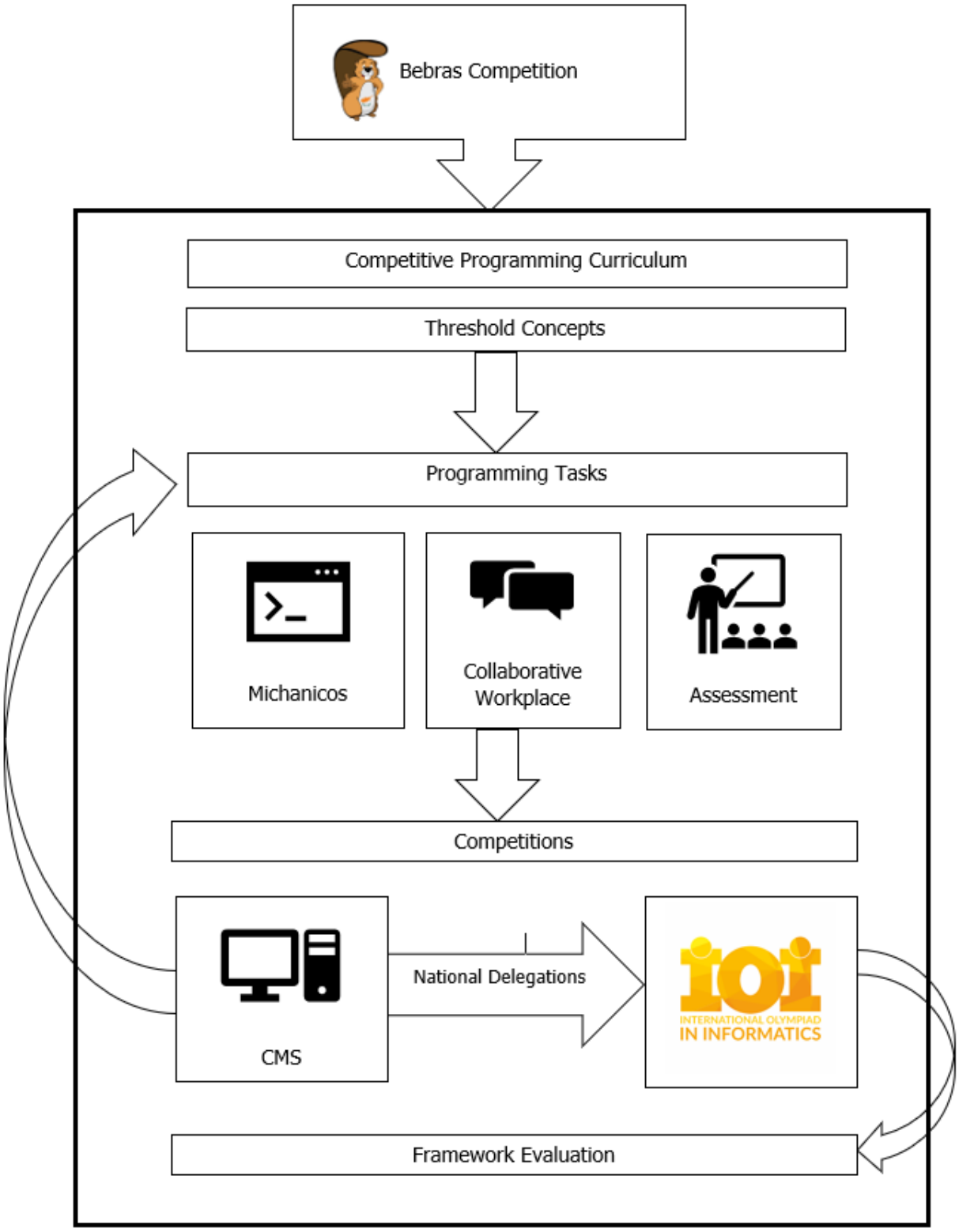


Figure 79: COI framework layout

Additionally, it reinforces the assigning and evaluation of tasks by automatically assessing them with real-time evaluation of submissions to both teachers and students.

Moreover, Michanicos provided the COI community with a competitive aspect. For students, this was established with the organisation of multiple programming competitions. The collaborative element for teachers was reinforced by coming together to create or assess tasks and design or modify the students' learning trajectory. Furthermore, I utilised the platform to evaluate students' performance. Michanicos has provided me with vital information for assessing the negotiations with liminality and identifying the places where students tend to get stuck.

To bring forth the ideas of putting together the critical components of the proposed framework and for successfully measuring the students' engagement and motivation with the learning process, I have used action research within the competition-type setting. Looking back, I cannot think of a more appropriate methodology applicable to this type of research project. The action research methodology has enabled me to navigate through liminality with my students, understand their struggles and range of emotions, acknowledge their feedback and make interventions where possible. The liminal space demands the introduction of adjustments explicitly as there is no universal order of topics or tasks that can guarantee the successful negotiation of the threshold concepts involved. The methodology has allowed me to collaborate with my colleagues and students in the most flexible way possible and explore their ideas and recommendations for improving the framework and its key components. We collectively tried to reach the same educational goals.

The action research methodology was very supportive since I was researching a situation with minimal data on new frameworks for competitive programming education. There is no universal proposal in the literature for a method or a system that has produced consistent results in IOI. Moreover, there has been inadequate data on identified threshold concepts in competitive programming compared to the number of studies for identifying threshold concepts in introductory programming courses. I believe that action research enhanced the educational/scientific consistency and the applicability factor of the COI

framework. This consistency is absent from many problem-solving methods and academic research studies in education today.

Furthermore, I have examined how the framework substantially impacted student performance by using the empirical data collected by the code-evaluation platform. I used the Python programming language and the SciPy package to analyse the correlation between perceived coding efficiency and actual performance on the threshold concept tasks. Additionally, I have measured the results from participating in local and international competitions with the threshold concepts embedded in programming tasks to provide a more accurate and impartial testament for the framework's validity. The progress has been evident from the improved performance in IOI competitions (Figure 80). Based on the total accumulated points and their average, COI students have demonstrated significant individual development between Japan and Azerbaijan (Tables 16-17). Expectantly, the constant improvement of the COI framework's components based on the collective feedback of both the COI and IOI communities can lead my country even higher.

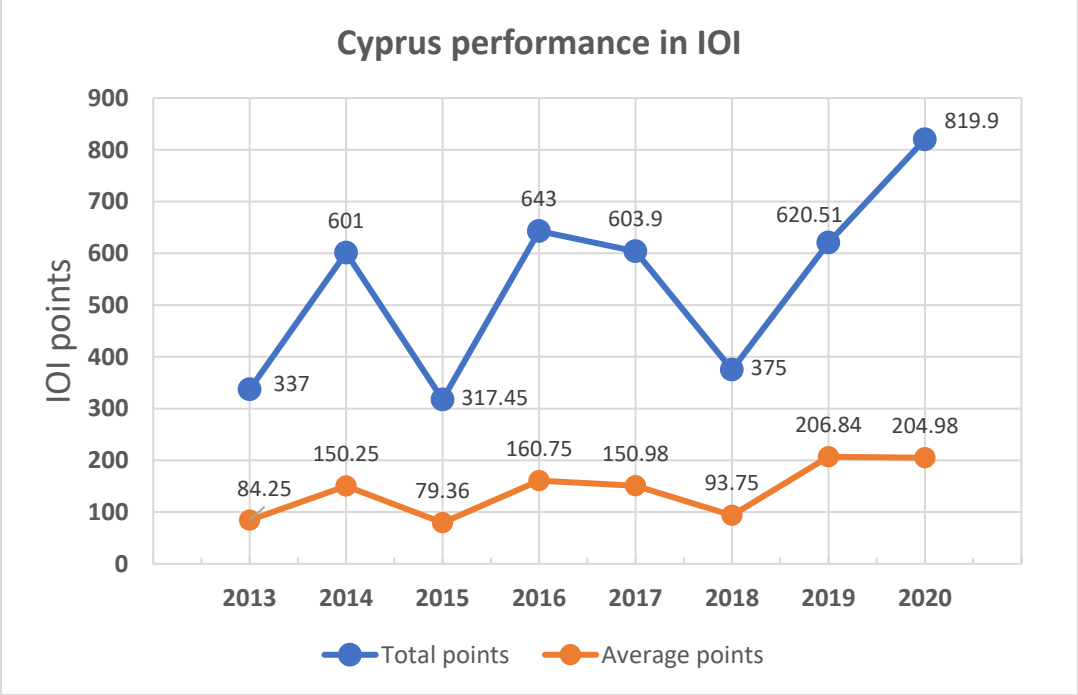


Figure 80: Performance of Cyprus teams in IOI competitions

Every teacher needs to improve their students' learning and quality of life. IOI medal winners can be admitted to the University of Cyprus without an entry exam as this is a directive of the MOEC to motivate the delegations. They also receive scholarships from top universities worldwide and job offers to work for global software companies. The Massachusetts Institute of Technology (MIT) is considered the 'holy grail' for every aspiring CS student. For some of my students, this is a dream that I have helped them turn into reality.

Since 2017, only four Cypriots have enrolled in MIT, all with a full scholarship. These four students have been a part of the COI community since they were 13 years old, and they have represented their country in numerous programming contests. One original contribution of this research is that it provides the stakeholders of my learning community with a theoretical and practical outline for competitive programming education and for preparing students for international competitions. To ensure continuation, I will continue to engage in research and evaluation, as I must guarantee that other students can accomplish similar achievements in the future.

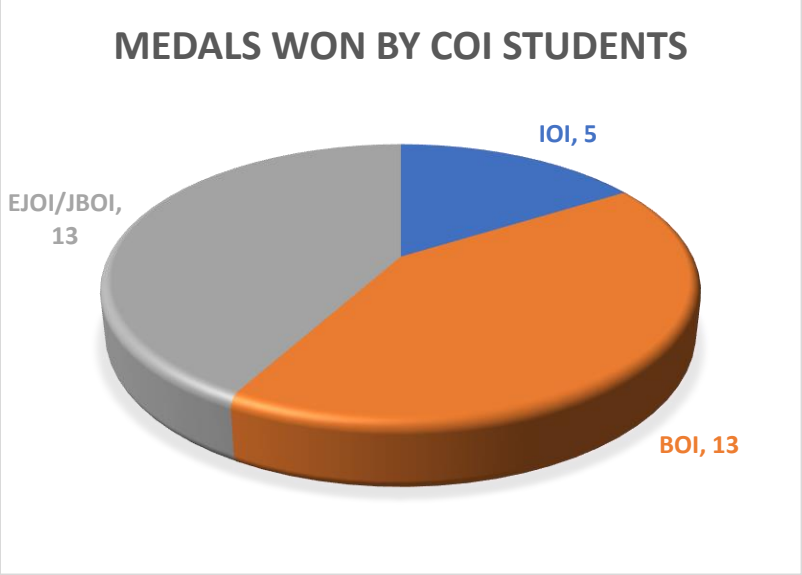


Figure 81: Medals won by COI students

Programming educators can discover additional research contributions in the code written for developing and testing the Michanicos platform, the complexity of the programming tasks created to challenge students, and lecture notes corresponding to university level

concepts. The exact degree of complexity of the proposed framework can be verified and confirmed upon implementation by IOI team leaders around the world.

The entire journey of my DProf studies was troublesome and lengthy. Looking back, I can identify all of the characteristics of a threshold concept within the research project itself. First, the project has been transformative as it provided me with a new way of viewing and describing it, and it has undoubtedly changed my perception of myself and the world around me. Second, the project has been irreversible as what I have learned and experienced cannot be unlearned or forgotten. Third, it has been integrative as it enabled me to bring together concepts and ideas that were previously unknown and unrelated. Fourth, it has been bounded as it clarified the scope of the community of learning and the field of practice. And last but not least, it has been very troublesome.

Troublesome not only in the sense of difficult to complete. I sometimes felt that the project study and my current mental schema were incompatible. I had experienced similar emotions of frustration and desperation as my students when they had to grasp the notions of dynamic programming and segment trees.

Furthermore, the duration of my negotiation with liminality seemed to be endless as one failure led to another. At times, the extent was overwhelming because I did not consider this as simply a DProf project. It was my life's work and a true testament to my contributions to learning and the lives of my students. Watching it get rejected repeatedly as a proposal made me feel that my work was not good enough and what I initially thought was a valid contribution to my field came crumbling down.

My perseverance and determination to finish what I started, a small indication of my pre-liminal space, allowed me to push through the negative aspects and concentrate on the positive. There were ups and downs during this study, and reflecting upon them reveals that the struggles were significantly more than the cheers. Completing this journey is as exhilarating as mastering a threshold concept, and the fact that I contributed to my community of learning makes it even more gratifying.

This empirical study introduced and assessed the COI framework that I have built based on the threshold concepts theory, the theory-based framework design by Dabbagh (2005), and investigated its potential implications for competitive programming. The research project has the potential to inform theory and practice for competitive programming education and offers a method that can produce consistent results in IOI (Figure 79). The study delivers three significant contributions to knowledge: the competitive programming threshold concepts, the methodology for identifying threshold concepts and assessing student performance in liminality, and the COI framework. The research findings are valuable for competitive programming educators, threshold concept researchers, and competitive programmers in secondary and tertiary education. At the end of this long journey, I am positive that I have contributed positively to my discipline and to the lives of my students, which makes this research worthwhile and fulfilling.

References

ACM/IEEE–CS Joint Task Force on Computing Curricula. (2013). Computer Science Curricula. USA: ACM Press and IEEE Computer Society Press.

Adjule Online Judge (2019). Available at: https://adjule.it/ (Accessed: 11 March 2019).

Akerlind, G., McKenzie, J., and Lupton, M. (2011). A threshold concepts focus to curriculum design: Supporting student learning through application of variation theory. Australian Learning and Teaching Council. Available at: http://www.olt.gov.au/system/files/resources/PP8_885_Final_Report_Akerlind_2011.pdf

Ala-Mutka, K.M. (2005). A survey of automated assessment approaches for programming assignments, Computer Science Education, 15(2), pp. 83-102. doi:10.1080/08993400500150747.

Ambrosio, A.P.L., and Costa, F.M. (2010). Evaluating the impact of PBL and tablet PCs in an algorithms and computer programming course. Proceedings of the 41st ACM Technical Symposium on Computer Science Education, SIGCSE 10. doi:10.1145/1734263.1734431.

Anderson, G. L., and Jones, F. (2000). Knowledge generation in educational administration from the inside-out: The promise and perils of site-based, administrator research. Educational Administration Quarterly, 36(3), pp. 428–464.

Apiola, M., and Tedre, M. (2012). New perspectives on the pedagogy of programming in a developing country context. Computer Science Education. 22, pp. 285-313.

Ariadne (2015). Available at: http://81.4.171.172/ariadni/login.php/ (Accessed: 7 November 2015).

Ashwin, A. (2008). What do students' examination answers reveal about threshold concept acquisition in the 14-19 age groups? In R. Land, J. H. F. Meyer, & J. Smith (eds.), Threshold concepts within the disciplines (pp. 173-184). Rotterdam: Sense Publications.

Atkinson, S. (1994). Rethinking the Principles and Practice of Action Research: The tensions for the teacher-researcher. Educational Action Research, 2(3), 383–401. https://doi.org/10.1080/0965079940020306.

Audrito, G., Demo, G., and Giovannetti, E. (2012). The role of contests in changing Informatics education: A local view, Olympiads in Informatics, 6.

Baldwin, L., and Kuljis, J. (2001). Learning Programming Using Program Visualization Techniques. Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 1, pp. 1051-1058

Balkan Olympiad in Informatics (2019). Available at: https://boi2019.epy.gr/ (Accessed: 30 September 2019).

Ball, S. (1977). Motivation in Education. Academic Press.

Barradell, S. (2012). The identification of threshold concepts: A review of theoretical complexities and methodological challenges. Higher Education, 65, pp. 265-276.

Barrows, H.S., and Tamblyn, R.M. (1980). Problem-based learning: An approach to medical education. Problem-Based Learning An Approach to Medical Education, pp. 1–17. Springer Publishing.

Bawamohiddin, A.B., and Razali, R. (2017). Problem-based Learning for Programming Education. International Journal on Advanced Science, Engineering and Information Technology, 7(6), pp. 2035–2050. doi:10.18517/ijaseit.7.6.2232.

Bebras Competition (2019). Available at: https://www.bebras.org/ (Accessed: 10 February 2019).

Bednar, A.K., Cunningham, D., Duffy, T.M., and Perry, J.D. (1991). Theory into practice: How do we link? In G. J. Anglin (Ed.), Instructional Technology: Past, present and future. Englewood, CO: Libraries Unlimited.

Bellman, R. (1954). The theory of dynamic programming. Bulletin of the American Mathematical Society, 60(6), pp. 503-516. doi:10.1090/s0002-9904-1954-09848-8.

Bellstrom, P., and Kilbrink, N. (2009). Problem-Based Learning in a Programming Context–Planning and Executing a Pilot Survey on Database Access in a Programming Language. In Information Systems Development, pp. 867-875. doi:10.1007/b137171.

Ben-Ari, M. (2001). Constructivism in Computer Science Education. Journal of Computers in Mathematics and Science Teaching, 20(1), pp. 45-73.

Bennedsen, J., and Caspersen, M. E. (2007). Failure rates in introductory programming. ACM SIGCSE Bulletin, 39(2), pp. 32–36.

Bergin, S., and Reilly, R. (2005). The influence of motivation and comfort-level on learning to program. Proceedings of the 17th Annual Workshop on the Psychology of Programming Interest Group. pp. 293-304,

Biggs, J. (2003). Aligning teaching and assessing to course objectives. Teaching and learning in higher education: New trends and innovations, 2, pp. 13-17.

Biggs, J.B., and Collis, K.F. (1982). Origin and description of the SOLO taxonomy. Evaluating the Quality of Learning. pp. 17-31. doi:10.1016/b978-0-12-097552-5.50007-7.

Biggs, J.B., and Tang, C.S. (2011). Teaching for quality learning at university: What the student does. Maidenhead: McGraw-Hill, Society for Research into Higher Education and Open University Press.

Bipp, T., Lepper, A., and Schmedding, D. (2008). Pair programming in software development teams – An empirical study of its benefits. Information and Software Technology, 50, 231–240. doi:10.1016/j.infsof.2007.05.006.

Bomia, L., Beluzo, L., Demeester, D., Elander, K., Johnson, M., and Sheldon, B. (1997) The impact of teaching strategies on intrinsic motivation. Champaign, IL: ERIC Clearinghouse on Elementary and Early Childhood Education.

Borzovs, J., Niedrite, L., and Solodovnikova, D. (2015). Computer Programming Aptitude Test as a Tool for Reducing Student Attrition. Environment. Technology. Resources. Proceedings of the International Scientific and Practical Conference, 3, 29. doi:10.17770/etr2015vol3.175.

Boud, D. (ed.) (1985). Problem-based Learning in Education/or the Professions. Sydney: Higher Education Research and Development Society of Australasia.

Boustedt, J., Eckerdal, A., Mccartney, R., Mostrom, J., Sanders, K., and Zander, C. (2007). Threshold concepts in Computer Science: Do they exist and are they useful? SIGCSE Bulletin, pp. 504-508. doi:10.1145/1227504.1227482.

Boyatzis, R.E. (1998). Transforming qualitative information: Thematic analysis and code development. Thousand Oaks, CA: Sage.

Breslow, L., Faye, A., Snover, L., and Masi, B. (2007). Methods of measuring learning outcomes and value added. Teaching and Learning Laboratory, Massachusetts Institute of Technology. Available at: https://tll.mit.edu/sites/default/files/guidelines/a-e-tools-methods-of-measuring-learning-outcomes-grid-2.pdf/ (Accessed: 6 May 2018).

Brodie, L., and Gibbings, P. (2007). Developing problem-based learning communities in virtual space. In R. Zehner & C. Reidsema (eds.), ConnectED 2007: International Conference on Design Education. University of New South Wales.

Brooks, R.E. (1990). Categories of programming knowledge and their application.International Journal of Man-Machine Studies, 33, pp. 241–246.

Brown, J., Collins, A., and Duguid, P. (1989). Situated cognition culture of learning. Educational Researcher, 18(1), 32.

Bruner, J. (1978). The Child's Concept of Language. New York Springer-Verlag.

Brush, T., and Saye, J. (2008). The effects of multimedia-supported Problem-based Inquiry on student engagement, empathy, and assumptions about History. Interdisciplinary Journal of Problem-Based Learning, 2(1). doi:10.7771/1541-5015.1052.

Bryman, A. (2001). Social Research Methods. New York: Oxford University Press.

Burton, B., and Hiron, M. (2008). Creating Informatics Olympiad tasks: Exploring the black art. Olympiads in Informatics, 2.

Butler, M., and Morgan, M. (2007). Learning challenges faced by novice programming students studying high level and low feedback concepts. In ICT: Providing choices for learners and learning. Available at: http://www.ascilite.org.au/conferences/singapore07/procs/butler.pdf

C++ shell (2019). Available at: http://cpp.sh/ (Accessed: 17 October 2019).

Carstensen, A.K., and Bernhard, J. (2008). Threshold concepts and keys to the portal of understanding: Some examples from electrical engineering. In R. Land, J. H. F. Meyer, & J. Smith (eds.), Threshold concepts within the disciplines (pp. 143-154). Rotterdam: Sense Publications.

Carter, J., and Jenkins, T. (1999). Gender and programming. ACM SIGCSE Bulletin, 31(3), pp. 1-4. doi:10.1145/384267.305824.

Caspersen, M. E., and Bennedsen, J. (2007). Instructional design of a programming course: A learning theoretic approach. Proceedings of the Third International Workshop on Computing Education Research, pp. 111–122. doi:10.1145/1288580.1288595

Chamillard, A., and Braun, K.A. (2000). Evaluating programming ability in an introductory computer science course. ACM SIGCSE Bulletin, 32(1), pp. 212–216.

Chamillard, A., and Joiner, J.K. (2001). Using lab practica to evaluate programming ability. ACM SIGCSE Bulletin, 33(1), pp. 159–163.

Cheang, B., Kurnia, A., Lim, A., and Oon, W. (2003). On automated grading of programming assignments in an academic institution. Computers and Education, 41, pp. 121-131.

Clear, T. (2001). Research paradigms and the nature of meaning and truth. ACM SIGCSE Bulletin, 33(2), pp. 9-10. doi:10.1145/571922.571930.

CMSocial (2019). Available at: https://github.com/algorithm-ninja/cmsocial/ (Accessed: 17 October 2019).

CodeBlocks (2019). Available at: http://www.codeblocks.org/ (Accessed: 17 October 2019).

Codeforces (2019). Available at: https://www.codeforces.com/ (Accessed: 17 October 2019).

Codility (2019). Available at: https://www.codility.com/ (Accessed: 17 October 2019).

Coghlan, D., and Brannick, T. (2005). Doing Action Research in your own organization. London: Sage.

Coleman, S. D., Perry, J. D., and Schwen, T. M. (1997). Constructivist instructional development: Reflecting on practice from an alternative paradigm. In Charles R. Dills and Alexander J. Romiszowski (eds.), Instructional Development Paradigms (pp. 269-282). Englewood Cliffs, NJ: Educational Technology Publications.

Collis, B., and Margaryan, A. (2005). Design criteria for work-based learning: Merrill's First Principles of Instruction expanded. British Journal of Educational Technology, 36(5), pp. 725–738. doi:10.1111/j.1467-8535.2005.00507.x

Combefis, S., and le Clement de Saint-Marcq, V. (2012). Teaching programming and algorithm design with Pythia, a web-based learning platform. Olympiads in Informatics, 6.

Combefis, S., and Wautelet, J. (2014). Programming training and Informatics teaching through online contests. Olympiads in Informatics, 8.

Connolly, T.M., and Begg, C.E. (2006). A Constructivist-Based Approach to Teaching Database Analysis and Design. Journal of Information Systems Education, 17(1).

Conole, G. (2008). Capturing practice: the role of mediating artefacts in learning design. In Handbook of Research on Learning Design and Learning Objects: Issues, Applications and Technologies, in L. Lockyer, S. Bennett, S. Agostinho, and B. Harper (eds), pp. 187-207, Hersey PA: IGI Global.

Conole, G. (2010). Review of pedagogical models and their use in e-learning. Computers and Education.

Costley, C., Elliott, G., & Gibbs, P. (2010). Doing work-based research: Approaches to enquiry for insider-researchers. SAGE. https://www.doi.org/10.4135/9781446287880.

Crescenzi, P., and Nocentini, C. (2007). Fully integrating algorithm visualization into a cs2 course.: a two-year experience. ACM SIGCSE Bulletin, 39(3), pp. 296-300. doi: 10.1145/1269900.1268869.

Creswell, J.W. (2012). Educational research: Planning, conducting, and evaluating quantitative and qualitative research. Boston: Pearson.

Creswell, J.W., and Plano Clark, V.L. (2011). Designing and conducting mixed methods research. Los Angeles: SAGE Publications.

Cutts, Q., Esper, S., and Simon, B. (2011). Computing as the 4th "R": a general education approach to computing education. In Sanders, K. (ed.) Proceedings of the Seventh International Workshop on Computing Education Research, pp. 133-138. doi:10.1145/2016911.2016938.

Cyprus Computer Society (2019). Available at: https://www.ccs.org.cy/ (Accessed: 10 July 2019)

Cyprus Olympiad in Informatics (2019). Available at: http://www.coinformatics.org/ (Accessed: 2 January 2019).

Dabbagh, N. (2005). Pedagogical models for E-Learning: A theory-based design framework. International Journal of Technology in Teaching and Learning, 1(1), pp. 25-44.

Dabbagh, N., and Bannan-Ritland, B. (2005). Online learning: Concepts, strategies, and application. Upper Saddle River, N.J.: Pearson, Merrill Prentice Hall.

Dagiene, V. (2005). Teaching Information Technology in general education: Challenges and perspectives. In Mittermeir R.T. (ed.) From computer literacy to Informatics fundamentals, pp. 53-64. doi:10.1007/978-3-540-31958-0_7.

Dagiene, V., and Skupas, B. (2011). Semi-automatic testing of program codes in the high school student maturity exam. Proceedings of the 12th International Conference on Computer Systems and Technologies, pp. 564-569. doi:10.1145/2023607.2023701.

Dagiene, V., and Skupiene, J. (2004). Learning by competitions: Olympiads in Informatics as a tool for training high-grade skills in programming. 2nd International Conference Information Technology: Research and Education ITRE, pp. 79-83. doi:10.1109/ITRE.2004.1393650.

Daly, C., and Waldron, J. (2004). Assessing the assessment of programming ability. ACM SIGCSE Bulletin, 36, pp. 210–213.

Daly, T. (2011). Minimizing to maximize: An initial attempt at teaching introductory programming using Alice. Journal of Computing Sciences in Colleges, 26(5), pp. 23-30.

Danic, M., Radosevic, D., and Orehovacki, T. (2011). Evaluation of student programming assignments in an online environment. Proceedings of the 22nd Center European Conference on Information and Intelligent Systems, Faculty of Organization and Informatics. Available at: https://www.academia.edu/1262203/Evaluation_of_Student_Programming_Assignments_in_Online_Environments (Accessed: 4 April 2018).

Davies, P. (2006). Threshold concepts: How can we recognise them? In (Meyer and Land, 2006).

Davies, P., and Mangan, J. (2008). Embedding threshold concepts: From theory to pedagogical principles to learning activities. In R. Land, J. H. F. Meyer, & J. Smith (eds.), Threshold concepts within the disciplines. Rotterdam: Sense.

Davies, S. (2008). The effects of emphasizing computational thinking in an introductory programming course. 38th Annual Frontiers in Education Conference, T2C-3-T2C-8. doi:10.1109/FIE.2008.4720362.

Davies, S.P. (1993). Models and theories of programming strategy. International Journal of Man-Machine Studies, 39(2), pp. 237–267. doi:10.1006/imms.1993.1061.

Davis, E.A., and Linn, M.C. (2000). Scaffolding students' knowledge integration: Prompts for reflection in KIE. International Journal of Science Education, 22, pp. 819–837.

Davis, G.A., and Rimm, S.B. (2004). Education of the gifted and talented. Boston, MA: Pearson.

Denscombe, M. (2010). The good research guide: for small-scale social research projects. New York: Open University Press.

Derri, V., and Pachta, M. (2007). Motor Skills and concepts acquisition and retention: A comparison between two styles of teaching. International Journal of Sport Science, 3(3), pp. 37-47.

Dewey, J. (1900). The school and society. Chicago: University of Chicago Press.

Dewey, J. (1903). Studies in logical theory. Chicago: University of Chicago Press.

Dewey, J. (1916). Democracy and education: An introduction to the philosophy of education. New York: MacMillan.

Dewey, J. (1929). The quest for certainty. New York: Minton.

Dewey, J. (1948). Education and the philosophic mind. New York: MacMillan.

Dijkstra, E.W. (1968). A constructive approach to the problem of programme correctness. BIT Numerical Mathematics, 8, pp. 174-186. doi:10.1007/BF01933419.

Diks, K., Kubica, M., and Stencel, K. (2007). Polish Olympiad in Informatics - 14 years of experience. Olympiads in Informatics, 1.

Dolmans, D.H.J.M., Loyens, S.M.M., Marcq, H., and Gijbels, D. (2016). Deep and surface learning in problem-based learning: A review of the literature. Advances in Health Sciences Education, 21(5), pp. 1087–1112. doi:10.1007/s10459-015-9645-6.

Driscoll, M.P. (1994). Psychology of learning for instruction. MA: Allyn and Bacon.

Du Boulay, B. (1986). Some difficulties of learning to program. Journal of Educational Computing Research, 2(1), pp. 57–73.

Du Boulay, B., O' Shea, T., and Monk, J. (1981). The black box inside the glass box: Presenting computing concepts to novices. International Journal of Man-Machine Studies, 14(3), pp. 237-249. doi:10.1016/S0020-7373(81)80056-9.

Duch, B.J., Groh, S.E., and Allen, D.E. (2001). The power of problem-based learning. Sterling, VA: Stylus Publishing, Inc.

Duffy, T.M., and Cunningham, D.J. (1996). Constructivism: Implications for the design and delivery of instruction. In D.H. Jonassen (Ed.), Handbook of educational communications and technology (pp. 170-198). New York: Simon & Schuster Macmillan.

Duffy, T.M., and Jonassen, D.H. (1992). Constructivism and the Technology of Instruction: A Conversion. Hillsdale, NJ: Erlbaum.

Dunkin, M., and Precians, R. (1992). Award-winning university teachers' concepts

Dyson, M., and Barreto-Campello, S. (2003). Evaluating Virtual Learning Environments: What are we measuring. Electronic Journal of E-Learning, 1, pp. 11–20.

Eckerdal, A. (2009). Novice Programming Students' Learning of Concepts and Practice. Doctoral dissertation, Acta Universitatis Upsaliensis.

Eckerdal, A., McCartney, R., Mostrom, J.E., Sanders, K., Thomas, L., and Zander, C. (2007) From limen to lumen: Computing students in liminal spaces. New York: ICER.

Edmodo. (2019). Available at: https://new.edmodo.com/ (Accessed: 15 December 2019).

Entwisle, N. (1998). Motivation and Approaches to Learning: Motivating and Conceptions of Teaching. In Motivating Students, S. Brown et al. (eds.), Kogan Page.

Eracleous, P. (2004). A proposal for the development of a three-year curriculum programme on Computer Science for secondary schools (Lyceums) of Cyprus. Assignment for MProf in Work-Based Learning Studies. Middlesex University.

Eracleous, P., Pavlikas, P., Ttofari, A., and Charalampous, A. (2019). Cyprus Olympiad in Informatics', Olympiads in Informatics. 13.

Ersoy, E., and Baser, N. (2014). The Effects of Problem-based Learning Method in Higher Education on Creative Thinking, Procedia-Social and Behavioral Sciences, pp. 3494-3498.

ESRC (2015) Framework for Research Ethics. Available at: https://unihub.mdx.ac.uk/__data/assets/pdf_file/0016/222730/framework-for-research-ethics_tcm8-33470.pdf/ (Accessed: 5 October 2018).

European Commission (2008). Digital Literacy European Commission working paper and recommendations from Digital Literacy High-Level Expert Group. Available at: https://www.ifap.ru/library/book386.pdf/ (Accessed: 5 June 2018).

European Commission (2013). Survey of Schools: ICT in Education. Available at: https://euc.ac.cy/https://ec.europa.eu/newsroom/dae/document.cfm?doc_id=1813/ (Accessed: 15 February 2018).

European Junior Olympiad in Informatics (2019). Available at: https://www.ejoi2019.si/ (Accessed: 5 September 2019).

European University of Cyprus (2019). Available at: https://euc.ac.cy/ (Accessed: 10 February 2019).

Evans, G.E., and Simkin, M.G. (1989). What best predicts computer proficiency? Communications of the ACM, 32(11), pp. 1322–1327. doi:10.1145/68814.68817.

Fee, S.B., and Holland-Minkley, A.M. (2010). Teaching computer science through problems, not solutions. Computer Science Education, 20, pp. 129-144.

Feilzer, M.Y. (2010). Doing mixed methods research pragmatically: Implications for the rediscovery of pragmatism as a research paradigm. Journal of Mixed Methods Research, 4(1), pp. 6-16. doi:10.1177/1558689809349691.

Felter, W., Ferreira, A., Rajamony, R., and Rubio, J.C. (2015). An updated performance comparison of virtual machines and Linux containers. IEEE International Symposium on Performance Analysis of Systems and Software, pp. 171-172. doi:10.1109/ispass.2015.7095802.

Firdyiwek, Y. (1999). Web-based courseware tools: Where is the pedagogy? Educational Technology, 39(1), pp. 29-34.

Foley, J. (1994). Key concepts in ELT: Scaffolding. ELT Journal, 48(1), pp. 101–102. doi:10.1093/elt/48.1.101.

Fonte, D., da Cruz, D., Gancarski, A.L., and Henriques, P.R. (2013). A flexible, dynamic system for automatic grading of programming exercises. In 2nd Symposium on Languages, Applications and Technologies, pp. 129-144.

Forisek, M. (2006). On the suitability of programming tasks for automated evaluation. Informatics in Education, 5(1), pp. 63-76.

Forisek, M. (2013). Pushing the boundary of programming contests. Olympiads in Informatics, 7.

Furlong, J., Barton, L., Miles, S., Whiting, C., and Whitty, G. (2000). Teacher Education in transition: Re-forming professionalism? Buckingham: Open University Press.

Gagne, R. M. (1973). Learning and Instructional Sequence. Review of Research in Education, 1(1), pp. 3–33. doi:10.3102/0091732X001001003.

Gance, S. (2002). Are constructivism and computer-based learning environments incompatible? Journal of the Association for History and Computing, 1.

Garcia-Mateos, G., and Fernandez-Aleman, J.L. (2009). Make learning fun with programming contests. In Z. Pan, A.D. Cheok, W. Muller, and A.E. Rhalibi (eds.), Transactions on Edutainment II, pp. 246-257. doi:10.1007/978-3-642-03270-7_17.

Garner, S. (2009). A quantitative study of a software tool that supports a part-complete solution method on learning outcomes. Journal of Information Technology Education, 8, pp. 285-310.

Garner, S., Haden, P., and Robins, A. (2005). My Program is Correct But it Doesn't Run: A Preliminary Investigation of Novice Programmers Problems. 42, pp. 173–180.

Gbollie, C., and Keamu, H.P. (2017). Student Academic Performance: The Role of Motivation, Strategies, and Perceived Factors Hindering Liberian Junior and Senior High School Students Learning. Education Research International, 2017, 1789084. doi:10.1155/2017/1789084.

Gentner, D. (2002). Mental Models, Psychology of. In N. Smelser & P.B. Bates (eds.), International Encyclopedia of the Social and Behavioral Sciences. Amsterdam: Elsevier Science, pp. 9683–9687.

Golan, R., Kyza, E.A., Reiser, B.J., and Edelson, D.C. (2002). Scaffolding the task of analyzing animal behavior with the Animal Landlord software. Paper presented at the Annual Meeting of the American Educational Research Association, New Orleans, LA.

Gomes, A., and Mendes, A.J. (2007). An environment to improve programming education. Proceedings of the 2007 International Conference on Computer Systems and Technologies, Rachev, B., Smrikarov, A. & Dimov, D. (eds.), 88, pp. 1-6.

Greening, T. (1999). Emerging constructivist forces in Computer Science education: Shaping a new future? In Computer Science Education in the 21st Century. doi: 10.1007/978-1-4612-1298-0_5.

Greeno, J. G. (2006). Learning in activity. In R. K. Sawyer (Ed.), The Cambridge handbook of the learning sciences (pp. 79-96). New York: Cambridge.

Greenwood, D.J., and Levin, M. (2007). Introduction to Action Research: Social Research for Social Change. California: Sage.

Guzdial, M. (2011). From Science to Engineering - Exploring the dual nature of Computing Education Research. Communications of the ACM, 54, pp. 37-39. doi:10.1145/1897816.1897831.

HackerEarth (2019). Available at: https://www.hackerearth.com/ (Accessed: 5 September 2019).

HackerRank (2019). Available at: https://www.hackerrank.com/ (Accessed: 5 September 2019).

Hadjerrouit, S. (2008). Towards a Blended Learning Model for Teaching and Learning Computer Programming: A Case Study. Informatics in Education.

Hagan, D., and Markham, S. (2000). Does it help to have some programming experience before beginning a computing degree program? ACM SIGCSE Bulletin, 32, pp. 25-28. doi:10.1145/353519.343063.

Hagan, D., Sheard, J., and Macdonald, I. (1997). Monitoring and evaluating a redesigned first-year programming course. ACM SIGCSE Bulletin, 29(3), pp. 37–39. doi:10.1145/268809.268832.

Haghighi, P. D., Sheard, J., Looi, C.K., Jonassen, D., and Ikeda, M. (2005). Summative computer programming assessment using both paper and computer. In ICCE, pp. 67–75.

Halim, S., and Halim, F. (2009). Competitive programming at the National University of Singapore. Department of Computer Science, The National University of Singapore. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.458.8600&rep=rep1&type=pdf/ (Accessed: 21 July 2018).

Hamm, S. B. (2016). A Foundation for Spatial Thinking: Towards a Threshold Concept Framework in GIScience and its Implications for STEM Education. The University of Waterloo.

Hanks, B., McDowell, C., Draper, D., and Krnjajic, M. (2004). Program quality with pair programming in CS1. Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education. pp. 176-180.

Hannafin, M.J. (1992). Emerging technologies, ISD, and learning environments: Critical perspectives. Educational Technology Research and Development, 40(1), pp. 49-63.

Hansen, M., and Borden, V. (2006). Using Action Research to support academic program improvement. New Directions for Institutional Research, 130, pp. 47-62. doi:10.1002/ir.179.

Hattie, J.A.C. (2012) Visible learning for teachers. London, UK: Routledge.

Hellenic Computing Olympiad (2019). Available at: http://hellenico.gr/ (Accessed 7 June 2019).

Helminen, J., Malmi, L., and Korhonen, A. (2009). Quick introduction to programming with an integrated code editor, automatic assessment and visual debugging tool - work in progress. In Proceedings of the 9th International Conference on Computing Education Research, Koli Calling, pp. 59-62.

Herr, K., and Anderson, G. (2005). The Action Research Dissertation: A Guide for Students and Faculty. Sage Publications, Inc. doi:10.4135/9781452226644.

Hickey, D.T., Moore, A.L., and Pellegrino, J.W. (2001). The motivational and academic consequences of elementary Mathematics environments: Do constructivist innovations and reforms make a difference? American Educational Research Journal, 38(3), pp. 611-652. doi:10.3102/00028312038003611.

Hill, G. J. (2016). Review of a problems–first approach to first-year undergraduate programming. In S. Kassel & B. Wu (eds.), Software Engineering Education Going Agile. Switzerland: Springer International Publishing, pp. 73–80.

Hine, G.S.C. (2013). The importance of Action Research in teacher education programs. In Teaching and learning in higher education: Western Australia's TL Forum, Issues in Educational Research, 23(2), pp. 151-163.

Hmelo-Silver, C.E. (2004). Problem-Based Learning: What and how do students learn? Educational Psychology Review, 16(3), pp. 235-266. doi:10.1023/B:EDPR.0000034022.16470.f3.

Hmelo-Silver, C.E. (2012). International Perspectives on Problem-based Learning: Contexts, Cultures, Challenges, and Adaptations. Interdisciplinary Journal of Problem-Based Learning, 6, pp.10-15.

Hmelo-Silver, C.E., and Ferrari, M. (1997). The problem-based learning tutorial: Cultivating higher-order thinking skills. Journal of the education of the gifted, 20(4), pp. 40l-422.

Hmelo-Silver, C.E., Duncan, R., and Chinn, C. (2007). Scaffolding and Achievement in Problem-Based and Inquiry Learning: A response to Kirschner, Sweller, and Clark. Educational Psychologist, 42(2), pp. 99-107. doi:10.1080/00461520701263368.

Holloway, M., Alpay, E., & Bull, A. (2010). A quantitative approach to identifying threshold concepts in engineering education. Engineering Education 2010: Inspiring the Next Generation of Engineers, EE 2010. Available at: https://epubs.surrey.ac.uk/837616/

Howe, K., and Berv, J. (2000). Constructing constructivism, epistemological and pedagogical. In D. C. Philips (Ed.), Constructivism in education: Opinions and second opinions on controversial issues. pp. 19-40: Chicago: The National Society for the study of Education.

Hung, C.Y., Chang, T. W., Yu, P.T., and Cheng, P.J. (2012). The Problem-Solving Skills and Learning Performance in Learning Multi-Touch Interactive Jigsaw Game using Digital Scaffolds. Los Alamitos, CA: IEEE Computer Society.

Iaydjiev, I. (2013). A pragmatic approach to social science. E-International Relations Students. Available at: http://www.e-ir.info/2013/03/01/a-pragmatic-approach-to-social-science/ (Accessed: 10 November 10, 2018).

Ibrahim, M. (2012). Thematic Analysis: A critical review of its process and evaluation. West East Journal of Social Sciences, 1(1), 9, pp. 39-47.

Ideone (2019). Available at: https://ideone.com/ (Accessed: 10 February 2019).

IEEEXtreme (2019). 24-hour Programming Competition. Available at: https://ieeextreme.org/ (Accessed: 19 November 2019).

Ihantola, P., Vihavainen, A., Ahadi, A., Butler, M., Borstle, J., Edwards, S.H., Isohanni, E., Korhonen, A., Petersen, A., Rivers, K., Rubio, M.A., Sheard, J., Skupas, B., Spacco, J., Szabo, C., and Toll, D. (2015). Educational data mining and learning analytics in programming: Literature review and case studies. ITiCSE-WGR. doi:10.1145/2858796.2858798.

Ilsche, T., Schuchart, J., Schone, R., and Hackenberg, D. (2015). Combining instrumentation and sampling for trace-based application performance analysis. In C. Niethammer, J. Gracia, A. Knupfer, M.M. Resch, and W.E. Nagel (eds.), Tools for high performance computing, pp. 123-136. doi:10.1007/978-3-319-16012-2_6.

International Olympiad in Informatics (2019). Available at: https://ioinformatics.org/ (Accessed: 5 September 2019).

International Olympiad in Informatics (2019). Task: Arranging Shoes. Available at: https://ioi2019.az/source/Tasks/Day1/Shoes/NGA.pdf

IOI Statistics (2019). Available at: http://stats.ioinformatics.org/ (Accessed: 8 October 2019).

IOI Syllabus (2017). Available at: http://ioi2017.org/files/ioi-syllabus-2017.pdf/ (Accessed: 15 May 2017).

ISSEP (2019) 12th International Conference on Informatics in Schools Situation, Evolution and Perspectives (ISSEP). Available at: http://cyprusconferences.org/issep2019/ (Accessed: 6 December 2019).

Ivala, E. , Gachago, D. , Condy, J., and Chigona, A. (2013). Enhancing Student Engagement with Their Studies: A Digital Storytelling Approach. Creative Education, 4, pp. 82-89. doi: 10.4236/ce.2013.410A012.

Ivankova, N. (2014). Mixed Methods Applications in Action Research: From Methods to Community Action. Thousand Oaks, California: SAGE Publications, Inc.

Ivankova, N.V., and Wingo, N. (2018). Applying Mixed Methods in Action Research: Methodological Potentials and Advantages. American Behavioral Scientist, 62(7), pp 978-997. doi:10.1177/0002764218772673.

Jacobson, N. (2000). Using on-computer exams to ensure beginning students' programming competency. ACM SIGCSE Bulletin, 32(4), pp. 53–56.

James, W. (1907). Pragmatism: A new name for some old ways of thinking. Cambridge, MA: Harvard University Press.

Jenkins, T. (2002). On the difficulty of learning to program. Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, pp. 53-58.

Jiang, Z., and Xu, X. (2019). Design and Implementation of Fill-in-the-blank Questions based on Open-Source Online Judge System. doi:10.2991/iccia-19.2019.10

Johnson, A.P. (2005). A short guide to Action Research. Boston, MA: Pearson.

Johnson, R.B., and Onwuegbuzie, A.J. (2004). Mixed methods research: A research paradigm whose time has come. Educational Researcher, 33(7), pp. 14-26.

Johnson-Laird, P.N. (1983). Mental models: Towards a cognitive science of language, inference, and consciousness. Cambridge, MA: Harvard University Press. Available at: https://hal.archives-ouvertes.fr/hal-00702919

Jonassen, D. (1999). Constructivist learning environments on the web: Engaging students in meaningful learning. EdTech, Educational Technology Conference and Exhibition: Thinking Schools, Learning Nation. Singapore: Ministry of Education.

Jonassen, D. (1999). Designing constructivist learning environments. In C. Reigeluth, (Ed.), Instructional-design theories and models: A new paradigm of instructional theory (pp. 215-239). University Park: Pennsylvania State University.

Jonassen, D. H., Howland, J. L., Moore, J. L., and Marra, R. M. (2003). Learning to Solve Problems with Technology: A Constructivist Perspective. Upper Saddle River, New Jersey: Merrill Prentice Hall.

Jonassen, D.H. (1991). Objectivism versus constructivism: Do we need a new philosophical paradigm? Educational Technology Research and Development, 39(3), pp. 5-14.

Jonassen, D.H., Grabinger, R.S., and Harris, N.D.C. (1991). Instructional strategies and tactics. Performance Improvement Quarterly, 3(2), pp. 29-47.

Kaila, E., Rajala, T., Laakso, M.J., and Salakoski, T. (2009). Effects, experiences and feedback from studies of a program visualization tool. Informatics in Education, 3.

Kallia, M., and Sentance, S. (2017). Computing teachers' perspectives on threshold concepts: Functions and procedural abstraction. In Proceedings of the 12th Workshop in Primary and Secondary Computing Education, pp. 15-24. doi:10.1145/3137065.3137085.

Karzunina, D., West, J., Maschiao da Costa, G., Philippou, G., and Gordon, S. (2019). The global skills gap in the 21st century. QS Quacquarelli Symonds. Available at: http://info.qs.com/rs/335-VIN-535/images/The%20Global%20Skills%20Gap%2021st%20Century.pdf/ (Accessed: 29 October 2019).

Kavitha, D.R.K., JalajaJayalakshmi, V., and Rassika, R. (2018). Collaborative learning in Computer Programming Courses using E-Learning Environments. International Journal of Pure and Applied Mathematics, 118(8), pp. 183-189.

Kavitha, R.K., JalajaJayalakshmi. V., and Kaarthiekheyan, V. (2017). Adoption of Knowledge Management Framework in Academic Setting. International Journal of Pure and Applied Mathematics, 116(12), pp. 77-85.

Kay, J., Barg, M., Fekete, A., Greening, T., Hollands, O., Kingston, J.H., and Crawford, K. (2000). Problem-Based Learning for foundation Computer Science courses. Computer Science Education, 10(2), pp. 109-128. doi:10.1076/0899-3408(200008)10%3A2%3B1-C%3BFT109.

Kember, D., and Kelly, M. (1993). Improving Teaching through Action Research. Green Guide No. 14. Campbelltown, NSW: Higher Education Research and Development Society of Australasia.

Kendall, M.G. (1955). Rank Correlation Methods. New York: Hafner Publishing.

Khalife, J.T. (2006). Threshold for the introduction of programming: Providing learners with a simple computer model. 28th International Conference on Information Technology Interfaces, pp. 71-76. doi:10.1109/ITI.2006.1708454.

Kirk, J., and Miller, L. (1986). Reliability and validity in qualitative research. London: Sage.

Knuth, D.E. (1974). Computer programming as an art. ACM 17, 12, pp. 667–673. doi:10.1145/361604.361612.

Koehler, M.J., Mishra, P., Kereluik, K., Shin, T.S., and Graham, C.R. (2014). The Technological Pedagogical Content Knowledge Framework. In J. M. Spector, M. D. Merrill, J. Elen, & M.J. Bishop (eds.), Handbook of Research on Educational Communications and Technology (pp. 101–111). Springer New York. doi:10.1007/978-1-4614-3185-5_9.

Konecki, M. (2014). Problems in Programming Education and Means of Their Improvement. In B. Katalinic (Ed.), DAAAM International Scientific Book, 1(13), pp. 459-470. doi:10.2507/daaam.scibook.2014.37.

Kotovsky, K. (2003). Problem Solving-large/small, hard/easy, conscious/nonconscious, problem-space/problem-solver: The issue of dichotomization. The Psychology of Problem Solving (pp. 374-384). UK: Cambridge University Press.

Koulouri, T., Lauria, S., and Macredie, R.D. (2014). Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches. ACM Transactions on Computing Education (TOCE), 14(4).

Kruger, J., and Dunning, D. (1999). Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments. Journal of Personality and Social Psychology, 77(6), pp. 1121-1134. doi:10.1037//0022-3514.77.6.1121.

Kurland, D.M., and Pea, R.D. (1985). Children's mental models of recursive LOGO programs. Journal of Educational Computing Research, 1(2), pp. 235-243.

Kurnia, A. (2001). Online Judge. School of Computing. The National University of Singapore.

Kurnia, A., Lim, A., and Cheang, B. (2001). Online Judge. Computers and Education, 36(4), pp. 299-315. doi:10.1016/S0360-1315(01)00018-5.

Laakso, M.J., Salakoski, T., Grandell, L., Qiu, X., Korhonen, A., and Malmi, L. (2005). Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2, Informatics in Education, 4.

Laevers, F. (2000). Forward to Basics! Deep-Level-Learning and the Experiential Approach. Early Years, 20(2), pp. 20-29. doi:10.1080/0957514000200203.

Lahtinen, E., Ala-Mutka, K., and Jarvinen, H.M. (2005). A study of the difficulties of novice programmers. ACM SIGCSE Bulletin, 37(3), pp. 14-18. doi:10.1145/1151954.1067453.

Lampert, M. (2001). Teaching problems and the problems of teaching. New Haven CT: Yale University Press.

Land, R., Cousin, G., Meyer, J.H.F., and Davies, P. (2005). Threshold concepts and troublesome knowledge: Implications for course design and evaluation. In Rust, C. (ed.) Improving student learning diversity and inclusivity. Oxford: OCSLD.

Land, R., Meyer, J.H.F., and Flanagan, M.T. (2016). Threshold concepts in practice. Rotterdam Boston Taipei: Sense Publishers.

Lave J., and Wenger E. (1991). Situated Learning - Legitimate Peripheral Participation, Cambridge: Cambridge University Press.

Lewin, K. (1946). Action research and minority problems. In Lewin, G.W. (ed.) Resolving social conflicts. New York: Harper and Row.

Lewin, K. (1948). Resolving social conflicts. New York: Harper and Rowe.

Linn, M.C., and Dalbey, J. (1989). Cognitive Consequences of Programming Instruction. In E. Soloway & J.C. Spohrer (eds.), Studying the Novice Programmer, pp. 57-81. Hillsdale, NJ: Lawrence Erlbaum.

Lisewski, B., and Joyce, P. (2003). Examining the five-stage e-moderating model: design and emergent practice in the learning technology profession. ALT-J 11(2), pp. 55-66.

Lister, R. (2010). Geek genes and bimodal grades. ACM Inroads, 1(3), pp. 16-17.

Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Mostrom, J.E., Sanders, K., Seppala, O., and Simon, B. (2004). A multinational study of reading and tracing skills in novice programmers. ACM SIGCSE Bulletin, 36(4), pp. 119-150.

Looi, H.C., and Seyal, A. (2014). Problem-based Learning: An Analysis of its Application to the Teaching of Programming. IPEDR, 70, pp. 69-75.

Lucas, U., and Mladenovic, R. (2006). Developing new 'world views': Threshold concepts in introductory accounting. In J. H. F. Meyer, & R. Land (eds.), Overcoming barriers to student understanding: Threshold concepts and troublesome knowledge (pp. 148-159). London: Routledge.

Lyons, A., and DeFranco, J. (2010). A Mixed-Methods Model for Educational Evaluation. The Humanistic Psychologist, 38(2), pp. 146-158. doi:10.1080/08873267.2010.485912.

Ma, L., Ferguson, J.D., Roper, M., Ross, I., and Wood, M. (2008). Using cognitive conflict and visualisation to improve mental models held by novice programmers. Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, pp. 342-346. doi:10.1145/1352135.1352253.

MacGregor, J., Smith, B.L., Tinto, V., and Levine, J. H. (1999). Learning about learning communities: Taking student learning seriously. Materials prepared for the National Resource Center for The First-Year Experience and Students in Transition Teleconference, Columbia, South Carolina.

Mackenzie, N., and Knipe, S. (2006). Research dilemmas: Paradigms, methods and methodology. Issues in Educational Research. Available at: http://www.iier.org.au/iier16/mackenzie.html/ (Accessed: 17 October 2017).

Maggiolo, S., and Mascellani, G. (2012). Introducing CMS: A Contest Management System. Olympiads in Informatics, 6.

Malmi, L., and Helminen, J. (2010). Jype - A program visualization and programming exercise tool for Python. Proceedings of the 5th International Symposium on Software Visualization, pp. 153-162. doi:10.1145/1879211.1879234.

Manev, K. (2008). Tasks on graphs. Olympiads in Informatics, 2.

Maxcy, S. (2003). Pragmatic threads in mixed methods research in the social sciences: The search for multiple modes of inquiry and the end of the philosophy of formalism. In

A. Tashakkori, and C. Teddlie (eds.), Handbook of mixed methods in social and behavioral research. Thousand Oaks, CA: Sage, pp. 51-89.

Maxwell, J. (2005). Qualitative research design: An interactive approach. London: Sage.

Mayer R.E. (1989). The psychology of how novices learn computer programming. In E. Soloway & J. C. Spohrer (eds.), Studying the Novice Programmer. pp. 129–159. Hillsdale, NJ: Lawrence Erlbaum.

Mayer R.E. (1989). The psychology of how novices learn computer programming. In E. Soloway & J. C. Spohrer (eds.), Studying the Novice Programmer (pp. 129–159). Hillsdale, NJ: Lawrence Erlbaum.

Mayer, R.E. (1985). Learning in complex domains: A cognitive analysis of computer programming. Psychology of learning and motivation, 19, pp. 89–130.

Mayes, T., and Freitas, S. (2004). Review of e-learning theories, frameworks and models. JISC e-learning models study report.

Mazlack, L.J. (1980). Identifying potential to acquire programming skill. Communications of the ACM, 23(1), pp. 14–17. doi:10.1145/358808.358811.

McCartney, R., Boustedt, J., Eckerdal, A., Mostrom, J.E., Sanders, K., Thomas, L., and Zander, C. (2009). Liminal spaces and learning computing', European Journal of Engineering Education. 34(4), pp. 383-391. doi:10.1080/03043790902989580.

McCauley, R., Grissom, S., Fitzgerald, S., and Murphy, L. (2015). Teaching and learning recursive programming: A review of the research literature. Computer Science Education, 25(1), pp. 37-66. doi:10.1080/08993408.2015.1033205.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In working group reports from ITiCSE on Innovation and technology in Computer Science Education, pp. 125-180. doi:10.1145/572133.572137.

McGettrick, A., Boyle, R., Ibbett, R., Lloyd, J., Lovegrove, L., and Mander, K. (2005). Grand challenges in computing education - A summary. The Computer Journal, 48, pp. 42-48. doi:10.1093/comjnl/bxh064.

McGowan, I. (2016). Towards a Theory-Based Design Framework for an Effective E-Learning Computer Programming Course. International Association for Development of the Information Society.

McKernan, J. (1991). Curriculum Action Research: a handbook of methods and resources for the reflective practitioner. London: Kogan Page.

McLean, J. (2009). Triggering engagement in SoTL through threshold concepts. International Journal for the Scholarship of Teaching and Learning, 3(2), pp. 1–5.

McLoughlin, C., and Oliver, R. (1999). Pedagogic roles and dynamics in telematics environments. In M. Selinger and J. Pearson (eds.), Telematics in Education: Trends and Issues (pp. 32-50). Kidlington, Oxford: Pergamon.

McNiff, J. (1988) Action Research: principles and practice. London: Macmillan.

McNiff, J., and Whitehead, J. (2006). All you need to know about Action Research. Thousand Oaks. CA: Sage Publications.

Mendes, A. J., Paquete, L., Cardoso, A., and Gomes, A. (2012). Increasing student commitment in introductory programming learning. In Frontiers in Education Conference. pp. 1–6. New York, NY: IEEE.

MENDO (2019). Available at: https://mendo.mk/ (Accessed: 5 September 2019)

Merrill, M. D. (2002). First principles of instruction. Educational Technology Research and Development. 50(3), pp. 43-59.

Meyer, J.H.F., and Land, R. (2003) 'Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines', Improving Student Learning - Ten Years On, pp. 412-424.

Meyer, J.H.F., and Land, R. (2005). Threshold concepts and troublesome knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning. Higher Education, 49, pp. 373-388. doi:10.1007/s10734-004-6779-5.

Meyer, J.H.F., and Land, R. (2006). Overcoming barriers to student understanding: threshold concepts and troublesome knowledge. Routledge: London and New York.

Meyer, J.H.F., and Land, R. (2007). Stop the conveyor belt, I want to get off. Times Higher Education Supplement. Available at: https://www.timeshighereducation.com/news/stop-the-conveyor-belt-i-want-to-get-off/90288.article/ (Accessed: 10 November 2018).

Meyer, J.H.F., and Shanahan, M. (2003). The troublesome nature of a threshold concept in Economics', in Meyer, J.H.F., and Land, R. (eds.), Overcoming barriers to student understanding: Threshold concepts and troublesome knowledge. Routledge: London and New York.

Michanicos Code-Evaluation Platform (2019). Available at: http://81.4.170.42:8980/training/ (Accessed: 1 October 2019).

Mills, G.E. (2011). Action Research: A guide for the teacher researcher. Boston: Pearson.

Moore, D.T. (2007). Analyzing learning at work: an interdisciplinary framework. Learning Inquiry, 1(3), pp. 175-188.

Morelock, M.J., and Feldman, D.H. (2003). Extreme precocity: Prodigies, savants and children of extraordinarily high IQ. In Colangelo, N., and Davis, G. (eds.), Handbook of gifted education. Needham Heights, MA: Allyn and Bacon, pp. 455-469.

Morgan, D. (2014). Integrating qualitative and quantitative methods: A pragmatic approach. London: SAGE Publications. doi:10.4135/9781544304533.

Mostrom, J.E., Boustedt, J., Eckerdal, A., McCartney, R., Sanders, K., Thomas, L., and Zander, C. (2009). Computer Science student transformations: Changes and causes. Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education, pp. 181-185. doi:10.1145/1562877.1562935.

Mudgett, D.R. (2014). Teaching and Learning in Technical IT Courses: Innovative Practices in Teaching Information Sciences and Technology: Experience Reports and Reflections, J.M. Carroll. Cham: Springer International Publishing.

Mwanza, D. (2002). Conceptualising work activity for CAL systems design. Journal of Computer Assisted Learning, 18(1), pp. 84-92.

Nemeth, A., and Laszlo, E. (2015). Online training and contests for Informatics contestants of secondary school age. Edukacja-Technika-Informatyka, 6(1), pp. 273-280.

Nikula, U., Gotel, O., and Kasurinen, J. (2011). A motivation guided holistic rehabilitation of the first programming course. ACM Transactions on Computing Education (TOCE), 11(4).

Norman, G.R., and Schmidt, H.G. (1992). The psychological basis of Problem-Based Learning: A review of the evidence. Academic Medicine: Journal of the Association of American Medical Colleges, 67(9), pp. 557-565. doi:10.1097/00001888-199209000-00002.

Nowicki, M., Matuszak, M., Kwiatkowska, A., Sysło, M., and Bała, P. (2013). Teaching secondary school students programming using distance learning: a case study. Available at: https://www.academia.edu/18581423/Teaching_secondary_school_students_programming_using_distance_learning_a_case_study/ (Accessed: 12 May 2018).

Nuutila, E., and Malmi, L. (2005). PBL and computer programming - seven steps method with adaptations. Computer Science Education, 15, pp. 123-142.

Nuutila, E., Torma, S., Kinnunen, P., and Malmi, L. (2008). Learning Programming with the PBL Method - Experiences on PBL Cases and Tutoring. In J. Bennedsen (ed.), Issues in Introductory Programming Courses (pp. 47–67). Berlin Heidelberg: Springer-Verlag.

O'Donnell, R. (2010). A critique of the threshold concept hypothesis and an application in economics. 164. Available at: http://www.finance.uts.edu.au/research/wpapers/wp164.pdf.

O'Grady, M. (2012). Practical Problem-Based Learning in Computing Education. ACM Transactions on Computing Education (TOCE), 12, p. 10.

Oliveira, A.M.C.A., Santos, S.C.D., and Garcia, V.C. (2013). PBL in Teaching Computing: An Overview of the last 15 years. Frontiers in Education Conference, pp. 123-142.

Osmond, J., Turner, A., and Land, R. (2008). Threshold concepts and spatial awareness in transport and product design. In R. Land, J. H. F. Meyer, & J. Smith (eds.), Threshold concepts within the disciplines (pp. 243-260). Rotterdam: Sense Publications.

Palumbo, D. (1990). Programming language/problem–solving research: A review of relevant issues. Review of Educational Research, 60(1), pp. 65-89.

PAME (2019). Available at: https://github.com/samartzidis/PAME/ (Accessed: 7 July 2019).

Pankov, P. (2008). Naturalness in tasks for Olympiads in Informatics. Olympiads in Informatics, 2.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. New York, USA: Basic Books, Inc.

Parker, V., Lieschke, G., and Giles, M. (2017). Ground-up-top down: A mixed method action research study aimed at normalising research in practice for nurses and midwives. BMC Nursing, 16. doi:10.1186/s12912-017-0249-8.

Pea, R.D., and Kurland, D.M. (1984). On the Cognitive Prerequisites of Learning Computer Programming. Technical Report No.18. New York, NY: Bank Street College of Education.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. (2007). A survey of literature on the teaching of introductory programming. ACM SIGCSE Bull, 39(4), pp. 204-223.

Peirce C.S. (1868). Some consequences of four incapacities. Journal of Speculative Philosophy, 2, pp. 140-157.

Peng, W. (2010). Practice and Experience in the Application of Problem-Based Learning in Computer Programming Course. International Conference on Educational and Information Technology.

Pereira, H.B. de B., Zebende, G. F., and Moret, M. A. (2010). Learning computer programming: Implementing a fractal in a Turing Machine. Computers & Education, 55(2), pp. 767-776. doi:10.1016/j.compedu.2010.03.009.

Perkins, D. (1999). The many faces of constructivism. Educational Leadership, 57(3), pp. 6-11.

Perkins, D. (2006). Constructivism and troublesome knowledge. In Meyer, J.H.F., and Land, R. (eds.), Overcoming barriers to student understanding: Threshold concepts and troublesome knowledge. Routledge: London and New York.

PKU online judge (2019). Available at: http://poj.org/ (Accessed: 5 September 2019).

Pohl, W. (2006). Computer Science contests for secondary school students: Approaches to classification. Informatics in Education, 5(1), pp. 125-132.

Preece, J. (2001). Sociability and usability in online communities: Determining and measuring success. Behaviour & Information Technology, 20(5), pp. 347-356. doi:10.1080/01449290110084683.

Pullen, M. (2001). The Network Workbench and Constructivism: Learning Protocols by Programming. Computer Science Education, 11(3), pp. 189–202.

Puntambekar, S., and Kolodner, J.L. (2005). Toward Implementing Distributed Scaffolding: Helping Students Learn Science from Design. Journal of Research in Science Teaching, 42(2), pp. 185–217. doi:10.1002/tea.20048.

Quinnell, R., and Thompson, R. (2010). Conceptual intersections: Re-viewing academic numeracy in the tertiary education sector as a threshold concept. In J. H. F. Meyer, R. Land, & C. Baillie (eds.), Threshold concepts and transformational learning (pp. 147-163). Rotterdam: Sense Publishers.

Rajala, T., Kaila, E., Linden, R., Kurvinen, E., Lokkila, E., and Laakso, M.J. (2016). Automatically assessed electronic exams in programming courses. In Proceedings of the Australasian computer science week multiconference (p. 11). ACM.

Razak, W.M.W.A., Baharom, S.A.S., Abdullah, Z., Hamdan, H., Aziz, N.U.A., and Anuar, A.I.M. (2019). Academic Performance of University Students: A Case in a Higher Learning Institution. KnE Social Sciences, pp. 1294-1304. doi:10.18502/kss.v3i13.4285.

Research Ethics (2019). Available at: https://unihub.mdx.ac.uk/study/spotlights/types/research-at-middlesex/research-ethics/ (Accessed: 17 September 2019).

Reynolds, G. (2010). The secret to great work is great play. Available at: https://www.presentationzen.com/presentationzen/2010/03/we-were-born-to-play-play-is-how-we-learn-and-develop-our-minds-and-our-bodies-and-its-also-how-we-express-ourselves-play.html

Richards, L., and Richards, T. (1991). The transformation of qualitative method: Computational paradigms and research processes. In Fielding N.G., and Lee R.M. (eds.), Using computers in qualitative research. London, UK: Sage, pp. 38-53.

Rist, R. S. (2004). Learning to Program: Schema Creation, Application, and Evaluation. In Fincher, S. & Petre, M., (eds.), Computer Science Education Research. London, UK: Taylor & Francis, pp. 175-195.

Ritchie, J., Lewis, J., Nicholls, C., and Ormston, R. (2013). Qualitative Research Practice: A Guide for Social Science Students and Researchers. Sage Publications, Inc.

Robins, A.V. (2010). Learning edge momentum: A new account of outcomes in CS1. Computer Science Education, 20(1), pp. 37-71. doi:10.1080/08993401003612167.

Robins, A. (2019). Novice Programmers and Introductory Programming. In S. Fincher & A. Robins (eds.), The Cambridge Handbook of Computing Education Research.

Cambridge Handbooks in Psychology, pp. 327-376. Cambridge: Cambridge University Press. doi:10.1017/9781108654555.013.

Robins, A.V., Haden, P., and Garner, S. (2006). Problem distributions in a CS1 course. In Proceedings of the Eighth Australasian Computing Education Conference, CRPIT, 52, pp. 165-173.

Robins, A.V., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer Science Education, 13(2), pp. 137-172.

Rogoff, B. (1994). Developing understanding of the idea of communities of learners. Mind, Culture, and Activity, 4, pp. 209-229.

Rountree, J., and Rountree, N. (2009). Issues regarding threshold concepts in Computer Science. Proceedings of the Eleventh Australasian Conference on Computing Education, 95, pp. 139-146. Available at: http://crpit.scem.westernsydney.edu.au/confpapers/CRPITV95Rountree.pdf/ (Accessed: 5 June 2018).

Rowbottom, D.P. (2007). Demystifying Threshold Concepts. Journal of Philosophy of Education, 41(2), pp. 263-270. doi:10.1111/j.1467-9752.2007.00554.x.

Salleh, S.M., Shukur, Z., and Judi, H.M. (2013). Analysis of research in programming teaching tools: An initial review. Procedia, Social and Behavioral Sciences, 103, pp. 127-135. doi:10.1016/j.sbspro.2013.10.317.

Sanders, K., and McCartney, R. (2016). Threshold concepts in computing: past, present, and future. Proceedings of the 16th Koli Calling International Conference on Computing Education Research, pp. 91-100. doi:10.1145/2999541.2999546.

Sarjoughian, H.S., and Zeigler, B.P. (1996). Abstraction mechanisms in discrete-event inductive modelling. Proceedings of the Winter Simulation Conference, pp. 748-755. doi:10.1145/256562.256803.

Savery, J.R. (2006). Overview of Problem-Based Learning: Definitions and distinctions', The Interdisciplinary Journal of Problem-based Learning, 1(1). pp. 9-20. doi:10.7771/1541-5015.1002.

Savery, J.R., and Duffy, T.M. (1995). Problem Based Learning: An instructional model and its constructivist framework. Educational Technology Publications, 35(5), pp. 31-37.

Savin-Baden, M. (2008). Liquid learning and troublesome spaces: Journeys from the threshold? In J.H.F. Meyer, & R. Land (eds.), Threshold concepts within the disciplines (pp. 75-88). Rotterdam: Sense Publishers.

Savin-Baden, M., and Major, C. (2013). Qualitative research: The essential guide to theory and practice. Routledge, London.

Scheja, M., and Pettersson, K. (2010). Transformation and contextualisation: Conceptualising students' conceptual understandings of threshold concepts in calculus. Higher Education, 59, pp. 221–241.

Schmidt, H. G., DeGrave, W. S., DeVolder, M. L., Moust, J. H. C., and Patel, V. L. (1989). Explanatory models in the processing of science text: The role of prior knowledge activation through small group discussion. Journal of Educational Psychology, 81, pp. 610-619.

Schulte, C., and Bennedsen, J. (2006). What do teachers teach in introductory programming? Proceedings of the Second International Workshop on Computing Education Research, pp. 17-28. doi:10.1145/1151588.1151593.

SciPy (2019). Available at: https://docs.scipy.org/ (Accessed: 5 September 2019).

Scott, M.J., and Ghinea, G. (2013). Educating programmers: A reflection on barriers to deliberate practice. The Higher Education Academy.

Shaffer, D. W., and Resnick, M. (1999). "Thick" authenticity: New media and authentic learning. Journal of Interactive Learning Research, 10(2), pp. 195-215.

Shanahan, M.P., and Meyer, J.H.F. (2006). The troublesome nature of a threshold concept in economics. In J.H.F. Meyer, & R. Land (eds.), Overcoming barriers to student understanding: Threshold concepts and troublesome knowledge (pp. 100-114). London: Routledge.

Shanahan, M.P., Foster, G., and Meyer, J.H.F. (2006). Operationalising a threshold concept in ergonomics: A pilot study using multiple-choice questions on opportunity cost. International Review of Economics Education, 2, pp. 29–57.

Shanahan, M.P., Foster, G., and Meyer, J.H.F. (2010). Threshold concepts and attrition in first-year economics. In J.H.F. Meyer, R. Land, & C. Baillie (eds.), Threshold concepts and transformational learning (pp. 207-226). Rotterdam: Sense Publishers.

Sheard, J., Carbone, A., D'Souza, D., and Hamilton, M. (2013). Assessment of programming: pedagogical foundations of exams. In Proceedings of the 18th ACM conference on innovation and technology in computer science education. ACM, pp. 141–146.

Sheard, J., Simon, Hamilton, M., and Lonnberg, J. (2009). Analysis of research into the teaching and learning of programming. Proceedings of the Fifth International Workshop on Computing Education Research Workshop, pp. 93-104. doi:10.1145/1584322.1584334.

Shinners-Kennedy, D. (2016). How not to identify threshold concepts. In Land R., Meyer J.H.F., and Flanagan M.T. (eds.), Threshold concepts in practice. Rotterdam Boston Taipei: Sense Publishers, pp. 253-267.

Shinners-Kennedy, D., and Fincher, S.A. (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. ACM, pp. 9–18. doi:10.1145/2493394.2493396

Shuell, T.J. (1980). Learning theory, instructional theory, and adaptation. In R.E. Snow, P.A. Federico, & W.E. Montague (eds.), Aptitude, Learning and Instruction (vol. 1, pp. 277-301). Hillsdale, NJ: Lawrence Erlbaum.

Simons, K., and Klein, J. (2007). The impact of scaffolding and student achievement levels in a problem-based learning environment. Instructional Science, 35(1), pp. 41-72. doi:10.1007/s11251-006-9002-5.

Skiena, S.S. (2008). The algorithm design manual. London: Springer.

Slack (2019). Available at: https://slack.com/ (Accessed: 5 September 2019).

Slinger, J. (2011). Threshold concepts in secondary geography education. Paper presented at The Geographical Association Annual Conference, Surrey, UK.

Sockalingam, N., and Schmidt, H.G. (2011). Characteristics of Problems for Problem-Based Learning: The Students' Perspective. Interdisciplinary Journal of Problem-Based Learning, 5, pp. 3-16.

Soloway, E. (1986). Learning to program = Learning to construct mechanisms and explanations. Communications of the ACM, 29(9), pp. 850-858. doi:10.1145/6592.6594.

Soloway, E., and Spohrer, J.C. (1989). Studying the Novice Programmer, Hillsdale, NJ: Lawrence Erlbaum.

Soloway, E., Ehrlich, K., Bonar, J., and Greenspan, J. (1983). What do novices know about programming? In B. Shneiderman & A. Badre (eds.), Directions in Human-Computer Interactions. Norwood NJ: Ablex, pp. 27-54.

Sorva, J. (2012). Visual program simulation in introductory programming education. Department of Computer Science and Engineering, Aalto University. Available at: http://lib.tkk.fi/Diss/2012/isbn9789526046266/ (Accessed: 20 April 2017).

Sorva, J., Karavirta, V., and Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. ACM: Transactions on Computing Education, 13(4), pp. 1-64. doi:10.1145/2490822.

Sphere Online Judge (2019). Available at: https://www.spoj.com/ (Accessed: 28 March 2019).

Spohrer, J.C., and Soloway, E. (1986). Alternatives to Construct-Based Programming Misconceptions. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 183-191. doi:10.1145/22627.22369.

Stringer, E. (2007). Action Research. London: Sage Publications.

Tang, S., Zou, L., and Liao, X. (2016). A Research on Online Judge Technology Based on MOOC Platform. DEStech Transactions on Engineering and Technology Research.

Taylor, C. E. (2008). Threshold concepts, troublesome knowledge and ways of thinking and practicing: Can we tell the difference in biology? In R. Land, J. H. F. Meyer, & J. Smith (eds.), Threshold concepts within the disciplines (pp. 185-195). Rotterdam: Sense Publishers.

Taylor-Powell, E. (1998). Questionnaire Design: Asking questions with a purpose. The University of Wisconsin, Cooperative Extension. Available at: http://www.wcasa.org/file_open.php?id=933/ (Accessed: 6 February 2017).

Teddlie, C., and Tashakkori, A. (2009). Foundations of mixed methods research. Thousand Oaks, California: Sage.

Thota, N., Berglund, A., and Clear, T. (2012). Illustration of paradigm pluralism in computing education research. Proceedings of the Fourteenth Australasian Computing Education Conference, pp. 103-112.

Timmermans, J. A. (2014). Identifying threshold concepts in the careers of educational developers. International Journal for Academic Development, 19(4), pp. 305-317. doi:10.1080/1360144X.2014.895731.

Timus Online Judge (2019). Available at: https://acm.timus.ru/ (Accessed: 10 February 2019).

Torp, L., and Sage, S. (2002). Problems as possibilities. Alexandria, VA: ASCD.

Tuckman, B.W., and Harper, B.E. (2012). Conducting Educational Research. Lanham, Md: Rowman and Littlefield Publishers.

Turner, V. (1969). The Ritual Process: Structure and Anti-Structure. New York: Aladine De Gruyter.

USA Computing Olympiad (2019). Available at: http://www.usaco.org/ (Accessed: 14 June 2019).

Utting, I., Tew, A. E., McCracken, M., Thomas, L., Bouvier, D., Frye, R., Paterson, J., Caspersen, M., Kolikant, Y., Sorva, J., and Wilusz, T. (2013). A fresh look at novice programmers' performance and their teachers' expectations. In Proceedings of the ITICSE Working Group Reports Conference on Innovation and Technology in Computer Science Education (pp. 15–32). New York, NY: ACM.

UVa Online Judge (2019). Available at: https://onlinejudge.org/ (Accessed: 21 January 2019).

Van Gorp, M.J., and Grissom, S. (2001). An Empirical Evaluation of Using Constructive Classroom Activities to Teach Introductory Programming. Computer Science Education, 11(3), pp. 247-260.

Van Merrienboer, J.J., and Paas, F. G. (1990). Automation and schema acquisition in learning elementary computer programming: Implications for the design of practice. Computers in Human Behavior, 6(3), pp. 273–289. doi:10.1016/0747-5632(90)90023-A

Verhoeff, T. (1990). Guidelines for producing a programming-contest problem set. Available at: https://www.win.tue.nl/~wstomv/publications/guidelines.html/ (Accessed: 24 September 2018).

Verhoeff, T. (1997). The role of competitions in education. Available at: https://www.researchgate.net/profile/Tom_Verhoeff/publication/228714944_The_role_of_competitions_in_education/links/00463519f239c9e9af000000/The-role-of-competitions-in-education.pdf/ (Accessed: 2 May 2016).

Verhoeff, T. (2013). Informatics everywhere: information and computation in society, science and technology. Olympiads in Informatics, 7.

Vihavainen, A., Vikberg, T., Luukkainen, M., and Partel, M. (2013). Scaffolding Students' Learning using Test My Code. ITiCSE 13, pp. 117-122.

Voigt, J., Bell, T., and Aspvall, B. (2010). Competition-style programming problems for computer science unplugged activities. Available at: http://www.cosc.canterbury.ac.nz/tim.bell/cseducation/papers/Voigt%20Bell%20Aspvall%202009%20CEDETEL.pdf/ (Accessed: 18 August 2018).

Walker, A., and Leary, H. (2009). A Problem Based Learning Meta Analysis: Differences Across Problem Types, Implementation Types, Disciplines, and Assessment Levels. Interdisciplinary Journal of Problem-Based Learning, 3, pp. 3-24.

Walker, G. (2013). A cognitive approach to threshold concepts. High Educ 65, 247–263 (2013). doi:10.1007/s10734-012-9541-4.

Wang, Y., Wang, X., Jiang, Y., Liang, Y., and Liu, Y. (2016). A code reviewer assignment model incorporating the competence differences and participant preferences. Foundations of Computing and Decision Sciences, 41(1), pp. 77-91. doi:10.1515/fcds-2016-0004.

Wasik, S., Antczak, M., Badura, J., Laskowski, A., and Sternal, T. (2016). A survey on online judge systems and their applications. ACM Computing Surveys, 51(1), pp. 1-34. doi:10.1145/3143560.

Watson, C., and Li, F.W. (2014). Failure rates in introductory programming revisited. In Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education. pp. 39–44. New York, NY: ACM.

Webster, B.F. (1996). The real software crisis: The shortage of top-notch programmers threatens to become the limiting factor in software development. Byte Magazine, 21, p. 218.

Wenger, E. (1998). Communities of Practice: Learning, Meaning, Identity. Cambridge: Cambridge University Press.

West, R.E., and Williams, G. (2018). I don't think that word means what you think it means: A proposed framework for defining learning communities. Educational

Technology Research and Development. Available at: https://link.springer.com/article/10.1007/s11423-017-9535-0/ (Accessed: 11 April 2019).

White, G., and Sivitanides, M. (2002). A theory of the relationships between cognitive requirements of computer programming languages and programmers' cognitive characteristics. Journal of Information Systems Education, 13(1), pp. 59-66.

Whitehead, J., and McNiff, J. (2006) Action Research: Living theory. London: Sage.

Widowski, D., and Eyferth, K. (1986). Comprehending and recalling computer programs of different structural and semantic complexity by experts and novices. In H. P. Willumeit, ed., Human Decision Making and Manual control. Amsterdam: North-Holland, Elsevier, pp. 267-275.

Wiedenbeck, S., LaBelle, D., and Kain, V.N.R. (2004) Factors affecting course outcomes in introductory programming. Proceedings of 16th Workshop of the Psychology of Programming Interest Group. Carlow, Ireland, pp. 97-110.

Wilson, B. (1996). Constructivist learning environments: Case studies in instructional design. Educational Technology Publications: New Jersey.

Wilson, B., and Ryder, M. (1996). Dynamic Learning Communities: An Alternative to Designed Instructional Systems. Available at: https://eric.ed.gov/?id=ED397847

Winslow, L.E. (1996). Programming pedagogy - A psychological overview. ACM SIGCSE Bulletin, 28(3), pp. 17-22.

Wood, D., Bruner, J., and Ross, G. (1976). The role of tutoring in problem-solving. Journal of Child Psychology and Psychiatry, 17, pp. 89-100.

World Economic Forum (2018). The Future of Jobs Report. Centre for the New Economy and Society. Available at: http://www3.weforum.org/docs/WEF_Future_of_Jobs_2018.pdf/ (Accessed: 18 April 2018).

Yacob, A., Saman, M.Y., and Yusoff, M.H. (2012). Constructivism learning theory for programming through an e-learning. In 6th International Conference on New Trends in Information Science and Service Science and Data Mining (ISSDM), pp. 639-643.

Yehezkel, C., Ben-Ari, M., and Dreyfus, T. (2005). Computer architecture and mental models. ACM Sigcse Bulletin. 37. pp. 101-105. doi:10.1145/1047344.1047390.

Zander, C., Boustedt, J., Eckerdal, A., McCartney, R., Mostrom, J. E., Ratcliffe, M., and Sanders, K., (2008). Threshold concepts in computer science: a multi-national investigation. In: R. Land, J.H.F.Meyer, and J. Smith (eds.), Threshold Concepts within the Disciplines. Rotterdam: Sense Publishers, pp. 105-118.

Zhao, C.M., and Kuh, G.D. (2004). Adding value: Learning communities and student engagement. Research in Higher Education, 45(2), pp. 115-138. doi:10.1023/B:RIHE.0000015692.88534.de.CrossRefGoogle Scholar.

Zhao, W., Zhang, W., He, Y., Xie, X., and Wen, J.R. (2018). Automatically Learning Topics and Difficulty Levels of Problems in Online Judge Systems. ACM Transactions on Information Systems, 36, pp. 1–33. doi:10.1145/3158670.

Zuber-Skerritt, O. (1992). Action Research in higher education: Examples and reflections. London: Kogan Page.

Appendices

1. Questionnaire 1: Bebras students
2. Questionnaire 2: COI Alumni
3. Questionnaire 3: COI Students
4. Interviews: IOI 2019 delegation
5. Power of Attorney
6. Parents Letter of Permission (EJOI 2019)
7. BOI 2019 task (Icarus)
8. Parent/Guardian Consent Form
9. Programming books I authored currently used in lyceum courses
10. Approval for research from Cyprus Computer Society
11. BOI 2016 task (Lefkaritika)
12. Bebras 2019 task (Spies)
13. Training systems from IOI participating countries
14. Certificate of participation in ISSEP / CSERC 2019
15. Organisation of seminars for the C++ programming language