What Computing Instructors Did Last Summer: Experiences and Lessons Learned

Tina Vrieler, Aletta Nylén and Åsa Cajander
Department of Information Technology
Uppsala University
tina.vrieler@it.uu.se, aletta.nylen@it.uu.se, asa.cajander@it.uu.se

***Abstract*— This paper studies the first computing education summer camp of its kind in Sweden. Semi-structured interviews were performed with six of the camp's instructors focusing on their teaching experiences in relation to the instructional content and the children. The instructors struggled with meeting the wide range in children's programming experience, suggesting a lack of pedagogical knowledge. The results also shed some light on the challenges and advantages related to recruitment, content, and teaching strategy. Awareness of the camp instructors' experiences enables computing instructors, in both formal and informal settings, to enhance the design of learning environments for children.**

Keywords—non-formal learning; computing summer camp; programming; children;

I. INTRODUCTION

Non-formal learning of computing and programming through e.g. "computer clubs" is not new. However, recent efforts to improve the relevance of science and technology to young people have lead to an increase in the visibility of these initiatives over the last few years. These efforts are often motivated by the need to attract young people to computing, to increase their knowledge of computing, and to change their attitudes towards computing subjects [1], [2]. Unrestrained from the bureaucracy and rigorous rules of education systems, informal outreach efforts in computing have spearheaded the introduction to computing education to K-12 students. In Sweden, a pilot summer camp in computing education with particular focus on computer programming was launched in the summer of 2016 with the goal of sparking enthusiasm for digital creation among children aged 12-16. Being the first computing summer camp of its kind in Sweden, much can be learned from this program that can inform policymakers, educators, and other stakeholders of similar initiatives. Therefore, the research question of this study is: What are the instructors' (engineering students from a Swedish research university) field experiences of the children, the content, and their interrelationship? Awareness of these experiences enables computing instructors, in both formal and informal settings, to enhance the design of learning environments for children.

II. BACKGROUND

Sweden, together with an increasing number of countries all around the world, are in the process of introducing computing and programming to the K-12 school curriculum [3], [4]. Research has shown that there are many arguments for introducing computing in the early school years including support for individual's development of computational literacy, broadening participation among women and under-represented minorities [5], and changing the stereotypes that computing is a "male territory", only for geeks, or that computing is boring [6], [7]. Additional argument for introducing computer science (CS) related subjects in K-12 education is to redress a declining interest in computing among young students, particularly girls. A lack of interest and experience in computing is likely to discourage young people from studying CS in higher education [8]. Evidence shows that an early exposure to computing could ignite young people's interest in CS related fields [9]–[11]. However, exposure alone is no guarantee to engage young people in CS and in particular programming. It is after all the pedagogy and not the medium nor the tools and their characteristics that make a difference in learning [12], [13]. A good learning experience is also important for the development of interest in CS.

While there seems to be many reasons to introduce computing and programming in K-12 education, little research has described the experience of CS instructors when introducing contemporary computing tools and resources (e.g. Scratch, 3D printing) to young learners. Prior research on computing education has primarily focused on programming pedagogy at the undergraduate level, although an interest in research on computing pedagogy in K-12 education is growing, as more and more countries introduce CS related subjects to their school curriculum [3]. As computing topics gain momentum in the K-12 curriculum, it has become increasingly important to better understand the experiences of the instructors when engaging children in computing and programming activities.

III. RELATED WORK

Several research studies report on the lessons learned from organizing informal learning activities in computing for children. Adams [11] of Calvin College and Georgia Tech's summer camp [14] used a project-based pedagogy where the students had to apply what they learnt during the camp to complete a project. A project could comprise creating a game, a movie, a music video or a story. A project showcase where the students could show their creations would conclude the summer camp.

Lakanen et al. [2] reports the lessons learned from five years of teaching five-day programming outreach course in Finland with the objective to familiarize young students (aged

978-1-5090-5920-1/17/$31.00 ©2017 IEEE

12-18) with computer science related (CS) concepts, as well as to increase their interests in CS related topics. Resembling Adam's work, Lakanen et al. used game development as contextualization for computing education. The difference, however, was that Lakanen et al. emphasized the connection between game development and real-world computing problems by using text-based programming language. To learn the basic programming concepts, the students in Lakanen et al.'s study followed an online tutorial and were encouraged to ask questions. Once the tutorial was done the students were supported to work independently with their games using available resources and searching the web. This proved to be too difficult for the novice students and more scaffolding efforts had to be put in place.

Doerschuk et al.'s [15] reports on the INSPIRED academies, which were two, one-day academies for middle school girls and underrepresented minorities. The instructional material emphasized learning using hands-on activities in small teams of two to three students. Visual programming languages Scratch and Lego NXT robots were used to expose the students to basic CS concepts. The students also got to create their own webpage. Evaluation showed that robotics was the most popular activity among the children, followed by web page development, and lastly Scratch. No explanation was given for the students' preference.

Gr8 (Great) Designs for Gr8 (Great) Girls [6] is a one-day outreach program that ran three times and was unique in the sense that it did not merely focus on improving middle-school girls understanding of CS but also aimed to improve their perception of CS career. The program introduced students to three different computing activities: programming animation using Scratch, programming Mad Libs word game using WING IDE (text-based), and human-computer interaction (designing an MP3 player). The study's inclusive approach to CS teaching created an attitude change among the students, increasing their likelihood of pursuing a career in CS.

The evaluation of the outreach efforts in [2], [6], [11], [14], [15] uniformly reported that students enjoyed learning about CS and posttest showed a positive change in attitudes. Although the outreach programs have diverse goals, they all used a pedagogy focused on design and creativity. The breadth or depth of the syllabus, as well as the use of different software and tools demonstrate the diversity of computing and the intention of the outreach programs.

In terms of programming language, Sanchez-Ruiz and Jamba [16] reported on elementary school experiment where 4th and 5th graders were introduced to textual programming and found that although the students enjoyed learning to code, they disliked programs not working, fixing errors and syntax. Similar observations were reported in Lakanen et al.'s study where the most common issues for students when learning textual programming were related to syntax (e.g. writing brackets and semicolons). An essential benefit of textual programming, however, is that it allows students to directly deal with the fundamentals of programming as used in the industry [17].

The advantage of visual programming environments is that they provide an environment without compile-time and syntax errors, which makes visual programming more accessible to young people and novices [18]. Other benefits of visual programming are reported in [19], [20], which found an improvement in students performance, retention, attitudes and beliefs about CS, as well as understanding of basic programming concepts. The main benefit of visual programming, however, seems to be the facilitation of making the abstract more concrete [21]. Disadvantages related to visual programming include challenges in transition to textual programming [22], and the lack of authenticity in giving the students real-world programming experience [2].

IV. METHODOLOGY

The director of the summer camp provided documents related to the camp's organization and connected the researcher to six (out of sixteen) camp instructors. Five of the six instructors who participated in the study were engineering students at a Swedish research university. One instructor had technical knowledge (developer). The instructors had different levels of experience in teaching programming to young children, which is common in informal learning settings where there is no requirement that instructors have formal teaching qualifications. The instructors were asked to sign a consent form prior to the interview, which described the research purpose, assured the confidentiality of identity, and included the principal researcher contact details.

The research question of this study is open-ended and aims to describe the experiences of the camp instructors. With this in mind, the qualitative research method, semi-structured interviews, was chosen. Interviews are a particularly suitable method, as this study aims to explore the instructor's experiences, as well as understand the significance or meaning attached to these experiences [23], [24].

The interviews were performed by the principal author and were focused on the instructors' teaching experiences in relation to the instructional content and the children. Five interviews were conducted face-to-face, while one interview was conducted online using Skype for logistical reasons. The interviews were conducted in Swedish and were translated to English for the article. Each interview lasted between 60 and 80 minutes and was recorded and transcribed. The interviews started with the participants describing the organization and objective of the camp. Then, the instructors were asked to reflect on their teaching practices, with particular focus on 1) how and why the instructional content was taught and how this, in turn, lead to desired learning outcomes, and; 2) how the children appropriated the content and responded to the teaching practices.

To analyze the participants' experiences in more detail, thematic coding analysis was used following the steps outlined by Robson and McCartan [24]. The principal researcher read and analyzed the interviews, looking for the different ways in which the instructors described the children, the content, the teaching practices, and their interplay. The coding tool ATLAS TI was used to organize the data and to connect the codes and themes with the excerpts from each transcript document. Data analysis was performed inductively where the codes and overarching themes emerged as the researcher repeatedly read through the data [24]. Seven categories, and two super-

categories, describing the experiences of the instructors are presented under 'Results'.

V. The Pilot Summer Camp

The camp lasted for six weeks; each week the camp hosted 48 new participants, which sums up to a total of 288 summer camp participants. Participant demographics are summerized in Table 1. The camp fee ranged between 32 € - 730 € per child per week proportional to the monthly income of the household. The majority of the children were from a high-income household. Most of the camp's participants had little experience in programming. The age of the children ranged from 11-16 years old with the majority being 13-14 years old. To ensure gender balance, available capacity in the summer camp was reserved for an equal number of girls and boys. This proved to be a necessity, as the boy's camp capacity filled up in a matter of hours while the girls' camp capacity required additional advertisement before they were filled. The extra effort to recruit girls paid off, as 46% of the camp's participants were girls.

Since the summer camp could only host a limited number of children each week, the organizers decided to advertise the summer camp through a limited number of channels. Advertising was done mostly in other informal computing education initiatives (e.g. after-school programs) or directly to the employees of the private sponsors. There were 16 instructors working (paid employment) at the summer camp. The instructors were either engineering students enrolled at a technical university or had technical knowledge (developer). To abate the stereotype of CS being more compatible with the male gender role, half of the instructors that were employed were female. Besides teaching at the summer camp, the instructors were also responsible for developing the instructional material that was used.

A. Camp Organisation

The camp was organized into five learning "tracks": 1) Maker-space 2) Scratch-studio 3) Game-studio 4) 3D-studio 5) App-studio. Track one, two, and three required no previous programming experience, track four and five required some programming experience (e.g. Scratch). Each track lasted a full week (Sunday–Friday) except for App-studio, which lasted two weeks. The children could freely choose which track they wanted to participate in. The tracks had at least a 6:1 child-instructor ratio.

A typical day at the summer camp would start with breakfast, after which a 3-hour long workshop follows. The first workshop ends at lunchtime. After a lunch break of three hours, a second 3-hour long workshop begins and ends by dinnertime. The evening activities/playtime consists of different indoor and outdoor activities. Some of the activities relates to computing, e.g. robot hunt and hackathon/game night. During the evening meetings, the children reflect on their day with the camp instructors, listen to music, and relax before bedtime.

TABLE I. Participant Demographics

Gender	Boys	54%
	Girls	46%
Age	15-16 years old	20%
	13-14 years old	62%
	11-12 yeard old	18%
Programming experience	No experience	24%
	Little experience	63%
	A lot of experience	13%

B. Instructional Content

The instructional content is described according to the teacher's manual and was confirmed by the instructors. The instructional content in all of the tracks was designed to facilitate the children's interests. It was more important that the children left the summer camp with a positive experience with computing and programming rather than achieving all of the learning objectives.

1) Maker-Space and Scratch-Studio

Scratch was taught as the first programming language both in the maker-space group and in the Scratch-studio group. The children worked independently to create simple games or stories following examples from the website 'kodboken.se' (an online learning resource for novice programmers). Coding the games or stories introduced the children to basic programming concepts such as variables, conditional statements, and functions. As the children were novices to computing and programming, it was determined that a lecture-style teaching, where the children were taught step-by-step, was the most appropriate. Once the children understood these concepts they were encouraged to create their own games independently or to follow a step-by-step guideline available on kodboken.se

The children in the maker-space group also got to work with microcontrollers (CodeBug and Arduino), computer-aided design (CAD) and 3D printing. The objective of using microcontrollers was to ignite children's interests in electronics, as well as to teach them how programming could be used to control electronic equipment. The children were introduced to how simple electrical circuits work and were encouraged to share examples of areas where electrical circuits are used.

Once the children were acquainted with how to use and program the microcontrollers, they worked in pairs to create a game controller, which they used to control their individual Scratch games. The children also built a game called "stable hands" using a microcontroller, which involves moving a metal loop along the length of a curved wire without touching the loop to the wire. If the loop touches the wire the microcontroller would display "Game Over", signaling that the game was lost. Building "stable hands" aided the children in understanding the difference between input and output data.

At the end of the microcontroller workshops, the children were able to connect simple electrical circuits containing LED lights, buttons, and resistors. Also, the children were able to create simple programs for the microcontroller using Scratch where they, for example, used buttons as input and LED lights as output.

The basic functions of CAD and 3D printing were taught to the children by showing them how to create simple shapes using operations such as subtraction, sweep, extrude and tweak. The children also learned how to import 3D files from different 3D printing communities, e.g. thingiverse and tinkercad. At the end of the workshop, the children could import and modify 3D designs, as well as print the design using the 3D printer. The children were encouraged to create and print their own designs.

On the last day of the camp the children worked on creating their own project inspired by what they had learnt during the week. The projects comprised of anything from a game in Scratch to a 3D printed volcano with LED lights. A showcase of all the projects concluded the week.

2) Game-Studio

Since most of the children were novice programmers, they were first introduced to basic programming concepts such as variables, script, operators, and loops by solving different programming exercises. Once the children reasonably could program they started working on developing their game, which was showcased at the end of the week to the rest of the camp. The children were encouraged to work in pairs.

The children's game projects were developed using a game engine called Defold and it's programming language Lua (a lightweight multi-paradigm programming language). The children learned how to use the game engine by playing the "Bug Game". The game gave the children access to the source code of which there were bugs that had to be fixed in order for the game to function. As the children fixed the bugs they learned some of the key concepts in game programming such as how to perform collision detection, capture input actions, change properties, display score and adding lives. The children also had access to video tutorials, which explained the different concepts.

The instructors also discussed with the children the theory and principles of game design, including the various components that makeup a game, the different categories of games (e.g. first person shooter, role-playing game), and the use of graphics, forms, and sounds in games. The social aspects of gaming were also discussed, including topics such as gamification in everyday life, the gamer identity, and gender representation and stereotypes in games.

3) 3D-Studio

The 3D-studio workshops were originally focused on 3D game programming but this focus changed during the course of the camp, as there was a mismatch between the children's level of programming skills and the skills required to create a 3D game using object-oriented programming language C#. Although the children were taught how to solve simple programming exercises (e.g. how to write output and create variables) the focus of their work was mainly on game design using the game engine Unity. As with game studio, the children were encouraged to work with a peer to come up with a design for a game, which they presented to the rest of the camp at the end of the week.

4) App-Studio

The children in the app-studio group had some programming skills and attended the camp for two consecutive weeks. During the first week, the focus of the workshops was on the children's understanding of how to program in a 2D mobile development platform called Corona SDK. Concepts such as functions, loops, variables, scaling, RGB, physics engine, and the coordinate system were taught to the children through the development of mobile games. Examples of games that the children developed included the classic game of Tic Tac Toe, fast click game (the game counts the number of clicks in a time interval), and balloon-popping game (balloon pops as the player clicks on them). The children worked in pairs or individually, depending on the level of difficulty of the exercises.

VI. Results

A. Understanding the Children and Their Individual Needs

1) The Children

The instructors mentioned several reasons why the children joined the summer camp, with the most common reason being encouragement from the parents. Instructor J5 and L6 believed that many of the children's parents worked in the technology sector, and therefore saw the importance of CS education for their child. Other reasons for participation in the summer camp, according to the instructors, included interest in programming, digital creation, and computer games; curiosity about programming and digital creation; and social reasons (hanging out with friends or meeting new friends). The majority of the children had tried programming in Scratch, although few had experience in textual programming. According to instructor P2, H3, S1, and J5 there was little diversity in terms of ethnicity and socio-economic status among the camp participants.

2) Meeting the Needs of the Children

The variation in the children's programming skills within each track made it particularly challenging for the instructors to meet the needs of individual children, to keep them motivated, and to ensure that all of the children achieved the learning objectives. *"It was hard to find their [the children's] level. If you start too high then they will lose interest and if you start too low they will also lose interest. It [the task] must be at a reasonable level for them so they can further develop their skills"* (Instructor L6). Instructor P2 discusses his/her feeling of inadequacy: *"There's a lot of running around trying to help a student and you don't have time to properly explain. You just want them to get started with their work so you can move on to the next student. (...) You want them to understand things correctly, even though it is not our focus – it is nice to think that they haven't understood it all wrong. I don't want them to leave thinking they have understood how it works and then it turns out they have no idea"*.

To better meet the children's diverse learning needs the instructors organized a coding studio for tracks two, three, four and five. Since the coding studio focused on textual programming, it was optional for the children in Scratch (track two) to participate. The children were divided into groups according to their programming skills. The coding studio replaced the three-hour workshops and focused on teaching the children basic elements of programming such as variables, strings, control structures, and loops. According to the instructors, grouping the children by their programming skills allowed the children to move at their own pace; the learners who progressed quickly were given more challenging tasks, and additional instructional support was given to the learners who progressed slower.

B. *Advantages and Disadvantages of Different Learning Contexts and Tools*

1) Using Games

All of the tracks except maker-space used games context to learn programming. The focus on games was justified by: 1) it was something that the children could relate to (since many youngsters play computer games); 2) the use of graphics in games motivate children and; 3) to show and inspire the children that it is easy to create fun games. Instructor P2 reflected on game programming and how it potentially could exclude girls: "*The way I see it, the risk with games is that it is only for stereotype gamers. (…) It would be a strong argument for us to introduce other types of digital media".* Instructor H3 also reflected on this matter but was perplexed as to what he/she could do: *"I think it [using game to learn programming] is a problem and I have thought about it a long time. All the examples we see contain game [elements], more or less: yeah, let's make a game! No, we should find a better example, which is really hard".*

2) Maker-Space

Maker-space was the only track that included several activities (Scratch, CodeBug, Arduino, CAD and 3D printing) and this diversity was reported as something that the children particularly appreciated. The instructors noticed that many girls had chosen maker-space and when asked what the reason for that might be, instructor S1 explained: *"I think partly it [maker-space] was a very creative track where you had to build things pretty much yourself, where you got to work with CAD, which was a little more practical. I also think that many girls have not tried programming before and since it was a beginner's track it suited them well to try out many different things".* An activity that was especially popular among the children in maker-space was 3D printing. Instructor A4 explained what he/she thought was the reason for this: *"I think it [3D printing] was about seeing something concrete. You drew something in the computer and out comes a piece of plastic. It is a cool technology, you have heard of it, you can relate to it. (…) And it's precisely this that you can take home something that I think is very appealing to them [the children], like: I have created this!".*

3) Visual vs. Textual Programming

Scratch was used as programming language for children in both the maker-space and Scratch-studio. The instructors reported that Scratch's mode of interaction, where only certain defined puzzle pieces could be interlocked, made the children less afraid of making mistakes. Overall, Scratch proved to be an easy and engaging programming language to learn for young children: *"I was surprised by Scratch, that I thought it was fun. In Scratch, they [the children] really got the opportunity to do whatever they wanted. They could often realize their ideas"* (H3, instructor). The instructors also appreciated that the children could easily follow how the code runs in Scratch and that the results of the programming work could be seen rapidly. Another benefit of Scratch, as reported by the instructors, is the plethora of featured projects and studios from the Scratch community (https://scratch.mit.edu/) that can inspire both educators and children in their learning process.

Programming language C# was used in the 3D-studio track but was found to be a difficult language for young learners: *"C# is not the most pedagogical language for beginners, a lot of curly brackets. And we noticed that the curly brackets killed them"* (P2, instructor). The instructors also mentioned other challenges associated with learning of textual programming, including lost of motivation due to the amount of time, effort, and focus to learn textual programming, and English comprehension problems. The programming language Lua, on the hand, was described as *"a pleasant language"* (H3, instructor), as the programming environment was simpler for the children to understand (e.g. curly brackets not required to enclose groups of statement).

4) Scaffolding the Transition from Visual to Textual Programming

Instructor P2, J5 and A4 noticed that it was challenging and frightening for many children to make the transition from visual to textual programming. *"When you [the children] started working in Scratch it can feel scary to move on to textual programming. You feel like you have complete understanding of how Scratch works and how the blocks work and you see the sequences. And then you start reading text, doesn't matter if it is Python, Lua or C#, it can sometimes feel overwhelming"* (P2, instructor). To smoothen the learning curve, as well as to address the children's anxiety to textual programming, the instructors observed that the 'Bug game' served this purpose well. 'Bug game' maintained the children's motivation by allowing the children to see graphics in programming all the while eliminating the need to understand programming concepts that required advanced mathematics. Instructor J5 explained: *"when the children started it [the Bug game], it wasn't working very well so they had to improve it. Instead of starting from an empty screen, which is quite a long way to go, and it would have taken three days before you could actually see things happening, that would have been boring I think. (…) This was a good way to attack the problem".*

5) The Creative Process as Motivator

A common denominator in all of the tracks was that the children enjoyed creating something of their own. Instructor J5 reported: *"Most of them [the children] thought that it was fun to do their own projects. (…) They worked in pairs and we gave them a paper where they had to write their project plan. What should we include in our game? Then we guided them and tried to create the game, and that was something many of*

them thought was fun because they could come up with anything they wanted". Encouraging children to come up with ideas, as well as realizing those ideas has been one of the guiding principles of the camp. Although many children were creative, there were children who were less inspired and motivated to learn. To engage those children, one of the instructors explained that it was particularly important to find what the children thought was fun and to provide different examples of learning activities. For example, instructor P2 explained that if the children did not like coding, they could design and build an obstacle course for a game instead. Once they were done with the course, they could test run their own and others' obstacle courses. But it was not always easy to motivate the children, as this was a pilot summer camp instructor P2 explained that some instructional content was developed ad-hoc to the problem at hand: *"it was very experimental the whole thing, the whole concept of having an IT summer camp with game tracks was new to us so the preparation was pretty much a gamble. It was like "we hope this works" and then we tested it"*.

VII. Lessons learned

This study set out with the aim to describe what the computing instructors learned from their field experiences of the student cohort, the content, and the interrelationship between the two. Although the pilot computing summer camp promotes gender equality there is still room for improvement when it comes to recruting a more diverse student group. One of the ways to increase diversity among the camp participants is to adopt a more inclusive recruitment approach in order to attract children that are not already interested or from groups that are underrepresented in CS. A more inclusive approach to promote the summer camp is planned for the subsequent year (2017), as studies show that the biggest impact of computing outreach efforts is on those children who were initially not interested or are underrepresented in CS [2], [14], [6].

Further, on the subject of promoting young people's interest in CS is the role of a young person's family. Data from the interviews indicates that parental support and CS qualifications are crucial when it comes to student engagement in CS. These results are consistent with those of Archer et al. [25] who highlighted the importance of family members' positive view and attitudes on children's science aspirations.

A constructivist approach to teaching was especially salient in project-based learning where children got to work on a problem of interest. The instructors reported that the children much enjoyed working on their own projects. A possible explanation for this might be simply due to the personal preferences of the children. Another possible explanation might be that the projects combined the versatility of computing with problem solving in a hands-on, creative, and collaborative manner. Research shows that these characteristics in a CS classroom are not only fun, but also inclusive in the sense that they attract a broader spectrum of learners [7].

Four of the tracks used computer games as context for learning to program. Although computer games can function as a constructive context to learn, it is arguably less versatile than maker-space. A too narrow focus on the technical aspects of CS can give the inaccurate message of CS being all about programming [7]. In addition, computer games has a tendency to attract only boys, as from the age of ten boys spend at least twice as much time playing computer games than girls [26]. It is important, therefore, that the range of programming examples displays the diversity of the field. This could be done e.g. by including programming of music video, webpage design, and interactive journalism (see [10]).

Scratch gave the summer camp participants an important first, positive learning experience with programming. The children were able to create advanced games and stories with relatively little effort. Scratch's gradual learning curve in programming could reduce the preconception of programming as being difficult and increase the children's self-efficacy in programming. Although many researchers [18], [27] promote the use of visual programming language in education to increase students' motivation and self-efficacy, there is risk for conceptual challenges when transitioning to textual programming [22]. In this study, the instructors observed that the transition between visual to textual programming was challenging for the children. This is in line with findings obtained by Powers et al. [22]. To "bridge" the transition gap, it is recommended to build on student's prior knowledge by e.g. using similar programming examples and cases in both visual and textual programming environments [28], [29].

Including scaffolding supports such as the 'Bug game' was found to be an appropriate way to introduce children to textual programming, as it reduces the complexity of different programming concepts. Previous research studies show that better learning performance requires relevant scaffolding [30], [31].

The experiences of the instructors reveal that the choice of text-based programming language for novice learners is crucial in creating a positive learning experience for the children. Programming languages that reduce the cognitive load on novices by e.g. simplifying syntax and semantics can maintain children's motivation and, may therefore be a more suitable choice.

One of the main challenges faced by the instructors was to meet the wide range in children's needs in order to create a positive learning experience. A possible explanation for this might be the instructors' lack of teaching experience and pedagogical understanding of what novice young learners need. Although the instructors had significant content knowledge, knowledge about good teaching in the domain, the so called pedagogical content knowledge (PCK), would have helped the instructors meet the needs of the learners better [5]. Involving licensed teachers in the camp to assist and complement the instructors with pedagogical knowledge could create a better learning experience and environment for both the children and the instructors.

VIII. The Validity of The Results

This study's findings have limitations that should be acknowledged. Although generalized claims are not the goal within the qualitative tradition it is recognized that the low number of participants in this study could reduce the credibility of the findings. However, the interview questions were designed to profoundly explore the experiences of the

instructors, and data collection ceased after the six interviews as no new themes or codes were gained from new interview data. The way the instructors were selected could also threaten validity, as it is likely that the instructors who wanted to participate in the study have a positive view of the camp. Since the instructors were employed by the camp they might be unwilling to disclose certain challenges and feel obligated to communicate only positive experiences. Although it is impossible for the researcher to detect the sincerity of the instructors' words, the instructors were guaranteed anonymity and were encouraged to speak their mind.

Another possible threat to validity is the way in which the data was interpreted. In qualitative research, the interpretation is "bound up with the 'self' of the researcher" [32, p. 305], and since only one researcher performed data analysis, it can be argued that the results of this study is subjective. On the other hand, the interpretation of the data was sent to the instructors for member checking, which arguably increases the findings' objectivity. To further counter the researcher bias, the researcher followed the recommendations of [24] and kept an open-mind during data analysis and actively considered alternative explanations for the findings.

IX. Conclusion and Future Research

This paper presented a detailed account of camp instructors' experiences in teaching CS to children. One of the main implications from this study is the importance of pedagogical content knowledge in the CS domain in order to meet the needs of the children. Effective teaching of CS goes beyond having adequate content knowledge. Instead, effective teaching is about understanding learners and their common misconceptions, as well as having the necessary tools and strategies to make a range of concepts comprehensible to different learners.

The second main implication of this study is the importance of the context in which programming is taught. If the aim is to broaden participation and make CS attractive to a diverse group of learners then the instructional content should reflect this diversity. Hence, it is vital to critically reflect on the choice of instructional content and the effects it might have on the learners. The results also shed some light on the challenges and advantages related to recruitment, content and teaching strategy. Taking these aspects into consideration, future research should examine how including this study's results in the CS classrom could affect the makeup of the student population, as well as student learning outcomes.

Acknowledgment

The authors would like to thank the instructors and the director of the summer camp for their invaluable contribution to this study. The authors would also like to thank Arnold Pears for proofreading.

References

[1] B. R. Maxim and B. S. Elenbogen, "Attracting K-12 Students to Study Computing," in *39th ASEE/IEEE Frontiers in Education Conference*, 2009, p. M1H–1–5.

[2] A.-J. Lakanen, V. Isomöttönen, and V. Lappalainen, *Five Years of Game Programming Outreach: Understanding Student Differences*. 2014, pp. 647–652.

[3] F. Heintz, L. Mannila, and T. Farnqvist, "A review of models for introducing computational thinking, computer science and computing in K-12 education," *2016 IEEE Front. Educ. Conf.*, pp. 1–9, 2016.

[4] Regeringskansliet (Swedish Government), "Stärkt digital kompetens i läroplaner och kursplaner," *09 March 2017*, 2017. [Online]. Available: http://www.regeringen.se/pressmeddelanden/2017/03/starkt-digital-kompetens-i-laroplaner-och-kursplaner/.

[5] M. Guzdial, *Learner-Centered Design of Computing Education: Research on Computing for Everyone*. Morgan & Claypool, 2016.

[6] M. Craig and D. Horton, "Gr8 designs for Gr8 girls: a middle-school program and its evaluation," *SIGCSE Bull.*, vol. 41, no. 1, pp. 221–225, 2009.

[7] J. Margolis and A. Fisher, *Unlocking the clubhouse: Women in Computing*. MIT Press, 2002.

[8] A. Pearl, M. Pollack, E. Riskin, B. Thomas, and A. Wu, "Becoming a Computer Scienctist," *Commun. ACM*, vol. 33, no. 11, pp. 47–57, 1990.

[9] M. Papastergiou, "Are Computer Science and Information Technology still masculine fields? High school students' perceptions and career choices," *Comput. Educ.*, vol. 51, no. 2, pp. 594–608, 2008.

[10] U. Wolz, M. Stone, S. Pulimood, and K. Pearson, "Computational thinking via interactive journalism in middle school," *ACM Tech. Symp. Comput. Sci. Educ.*, pp. 239–243, 2010.

[11] J. C. Adams, "Scratching Middle Sschoolers' Creative Itch," *Proc. 41st ACM Tech. Symp. Comput. Sci. Educ.*, p. 356, 2010.

[12] R. E. Clark, "Reconsidering Research on Learning from Media," *Rev. Educ. Res.*, vol. 53, no. 4, pp. 445–459, 1983.

[13] V. Garneli, "Instructional media and teaching methods for engaging children with computer programming," *Proc. - IEEE 14th Int. Conf. Adv. Learn. Technol. ICALT 2014*, pp. 768–770, 2014.

[14] B. Ericson and T. McKlin, "Effective and Sustainable Computing Summer Camps," *Proc. 43rd ACM Tech. Symp. Comput. Sci. Educ.*, pp. 289–294, 2012.

[15] P. Doerschuk, J. Liu, and J. Mann, "INSPIRED Computing Academies for Middle School Students: Lessons Learned," in *The Fifth Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiatives, Insight, and Innovations*, 2009, pp. 52–57.

[16] A. J. Sánchez-Ruíz and L. A. Jamba, "FunFonts: Introducing 4 th and 5 th Graders to Programming Using Squeak," in *ACM-SE 46_ Proceedings of the 46th Annual Southeast Regional Conference on XX*, 2008, pp. 24–29.

[17] V. Isomöttönen, A.-J. Lakanen, and V. Lappalainen, "K-12 Game Programming Course Concept Using Textual Programming," in *42nd ACM technical symposium on Computer science education*, 2011, pp. 459–464.

[18] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Habits of programming in scratch," in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE '11*, 2011, p. 168.

[19] T.-C. Wang, W.-H. Mei, S.-L. Lin, S.-K. Chiu, and J. M.-C. Lin, "Teaching programming concepts to high school students with Alice," in *2009 39th IEEE Frontiers in Education Conference*, 2009, pp. 1–6.

[20] B. Moskal, D. Lurie, S. Cooper, B. Moskal, D. Lurie, and S. Cooper, "Evaluating the effectiveness of a new instructional approach," *ACM SIGCSE Bull.*, vol. 36, no. 1, p. 75, Mar. 2004.

[21] W. Dann and S. Cooper, "Education: Alice 3: concrete to abstract," *Communications of the ACM*, ACM, 2009.

[22] K. Powers, S. Ecott, and L. Hirshfield, "Through the looking glass: Teaching CS0 with Alice," *SIGCSE 2007 38th SIGCSE Tech. Symp. Comput. Sci. Educ.*, vol. 1, pp. 213–217, 2007.

[23] A. James, M. Waring, R. Coe, and L. V Hedges, *Research methods and methodologies in education*. London: SAGE Publications, 2012.

[24] C. Robson and K. McCartan, *Real World Research*. Wiley, 2015.

[25] L. Archer, E. Dawson, J. DeWitt, A. Seakins, and B. Wong, "'Science capital': A conceptual, methodological, and empirical argument for extending bourdieusian notions of capital beyond the arts," *J. Res. Sci. Teach.*, vol. 52, no. 7, pp. 922–948, 2015.

[26] Statens Medieråd, “Ungar & medier 2015: fakta om barns och ungas användning och upplevelser av medier,” 2015.

[27] M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari, “From Scratch to ‘Real’ Programming,” *ACM Trans. Comput. Educ.*, vol. 14, no. 4, pp. 1–15, Feb. 2015.

[28] N. Tabet and H. Alshikhabobakr, “From Alice to Python . Introducing Text-based Programming in Middle Schools .,” *Proc. 2016 ACM Conf. Innov. Technol. Comput. Sci. Educ.*, pp. 124–129, 2016.

[29] D. N. Perkins and G. Salomon, “Teaching for Transfer,” *Educ. Leadersh.*, vol. 46, no. 1, pp. 22–32, 1988.

[30] J. S. Bruner, *Actual minds, possible worlds*. Harvard University Press, 1986.

[31] L. S. Vygotsky, *Mind and Society: The Development of Higher Psychological Processes*. Cambridge, MA, 1978.

[32] M. Denscombe, *The Good Research Guide: For small-scale social research projects*, Fourth Edi. Open University Press, 2010.