Informatics in Education, 2026, Vol. 25, No. 1, 137–172

DOI: https://doi.org/10.15388/infedu.2507.015

Three Programs, Three Years, and Four Concepts: Teachers' Views on Indirection, References, Scope, and Parameter Passing in CSED

Pontus HAGLUND[1,*] [000 ORCID 0000] , Linda MANNILA[1,2] [000 ORCID 0000] , Filip STRÖMBÄCK[1] [000 ORCID 0000] , Aseel BERGLUND[1] [000 ORCID 0000]

[1]Linköping University, Department of Computer and Information Science, Sweden

[2]University of Helsinki, Department of Computer Science, Finland

e-mail: pontus.haglund@liu.se, linda.mannila@liu.se, filip.stromback@liu.se, aseel.berglund@liu.se

Abstract. Computer science (CS) students are expected to grasp numerous CS concepts during their CS education. Researchers have previously pointed to some concepts that are challenging for many students to conquer during their education. In this study, we investigate how CS students encounter indirection, scope, references, and parameter transfer during their studies. We focus on the first three study years, as previous studies have indicated that students do not significantly improve their grasp of these concepts during that time. We surveyed the teachers of courses in three CS study programs, exploring teachers' perspectives on students' knowledge of the concepts and how explicitly the concepts are taught and graded. Our investigation highlights several ways in which curricula diverge from previous recommendations and how an understanding of these study programs can support learning outcomes.

Key words: Subtle Concepts, CSED, Parameter Passing, Indirection, Scope, References, Teachers' Perceptions.

1. Introduction

Students in Computer Science (CS) programs encounter a number of subtle but important concepts, such as indirection, references, scope, and parameter transfer. In this paper, we refer to these four concepts as subtle concepts. Subtle concepts are important, since they play an important role for students when acquiring other concepts, such as abstraction and concurrency (Strömbäck et al., 2019, 2023). Previous research has recognized ongoing challenges concerning students' ability to grasp and assimilate these concepts over the course of their academic journey (Strömbäck et al., 2023). Further, teachers' awareness of students' misconceptions and what students know and have experienced beforehand are

*Corresponding author.

important factors to consider when trying to deal with misconceptions (Qian and Lehman,2017). Previous research (Fisler et al., 2017) has proposed that subtle concepts need to be taught consistently and explicitly throughout a study program, raising questions about how students encounter these concepts across a program and how instructors perceive how students encounter these concepts across a program and how instructors perceive students' development in relation to them.

Qian and Lehman (2017) call for research to shift focus from documenting misconceptions to developing students' knowledge. Instructors play an important role in shaping students' learning: they define course expectations, design activities, and make decisions about what is assessed explicitly or implicitly. Yet relatively little is known about how instructors perceive students' mastery of these concepts, the level of mastery required by assignments, or the degree to which the concepts are reinforced across multiple courses. Understanding these perceptions can help identify areas where teaching practices, curricular structures, or assessment approaches might benefit from closer examination. The aim of this paper is therefore to approach students' misconceptions of subtle concepts from the perspective of instructors. We want to provide an overview of how teachers perceive students' proficiency in these concepts in their courses. Through this study, we aim to explore to what extent each teacher thinks that students are taught these concepts during the first three years of their studies.

We aim to gain a clearer understanding of how instructors perceive: 1) students' prerequisite skills related to the four subtle concepts, 2) the extent to which these concepts are taught and assessed within their courses, and 3) how these perceptions relate to the learning outcomes the courses are intended to support. We conduct our initial investigation by selecting three CS programs in which no significant improvement in students' grasp of these concepts has previously been observed (Strömbäck et al., 2023) and targeting the courses in those programs where programming plays a central role in achieving the learning objectives. We then survey instructors responsible for these courses about their impressions of how scope, parameter passing, aliasing, and indirection are addressed in their courses. By analyzing these self-reported perceptions, we do not endeavor to measure effectiveness: instead we explore where closer examination is needed by answering the following research questions:

> **RQ1** How do these CS programs align with previously established pedagogical recommendations for teaching subtle concepts (e.g., teaching them consistently and explicitly throughout eduction)?
>
> **RQ2** How can this type of overview serve to support the acquisition of concepts over a longer period of time than a single course?

The remainder of this paper is organized as follows: The next section presents the Background (Section 2), providing an overview of relevant literature concerning subtle concepts, constructive alignment, misconceptions, and problematic knowledge, as well as strategies for addressing such misconceptions. The Method section (Section 3) describes the CS programs examined, the criteria for course selection, and data collection procedures.

It also describes the statistical analyses conducted and the validity and reliability of the approach. The Results section (Section 4) reports the results of the investigation,followed by the Discussion (Section 5) and Conclusion (Section 6), which interpret the results in relation to existing research and consider their implications for the design and delivery of CS education.

2. Background

2.1 Subtle Concepts

Different studies have proposed varying definitions of the concepts relevant to this work (e.g. Fisler et al. (2017); Nelson et al. (2020)). The definitions below represent how these concepts are interpreted in the context of this study.

Indirection - manipulating or accessing values through the use of pointers and dynamic dispatch of function calls.

References - using aliases to manipulate or access values.

Both indirection and references are concepts centered around the aliasing of data. In this paper, indirection and references is used to give instructors the opportunity to distinguish between lower level aliasing through pointers (indirection) and a higher level aliasing (references). In languages such as C++ both higher and lower level aliasing is distinguishable both semantically and syntactically. While in other languages, such as Python, they are indistinguishable.

Scope - the scope of name binding.

This concept involves students grasping the visibility and lifetime of variables and functions (Nelson et al., 2020). in this paper we consider scope in relation to all semantic structures that affect lifetime and visibility (functions, objects, blocks, etc.).

Parameter Passing - transfer of parameters to functions, including applicable qualifiers.

This concept involves students grasping the semantic difference between call by reference and call by value (Nelson et al., 2020). It also involves students having the ability to work with parameters to pass data into functions. In this paper, return values are also considered part of this concept.

Indirection, references, scope and parameter passing were selected because they are foundational yet persistently challenging aspects of programming for students (Strömbäck et al., 2023). They operate at the intersection between syntax, semantics, and program behavior. These concepts have also been described as important prerequisites when approaching more advanced topics such as abstraction and concurrency (Strömbäck et al., 2019, 2023). Examining how instructors perceive and teach these concepts thus offers valuable insight into how core programming knowledge is constructed and reinforced across the curriculum.

The delineation used for the four concepts in this study is informed by the context of the curricula of the programs investigated. Students in these programs engage withprogramming languages that include both explicit pointer manipulation and references, such as C++. They also engage with languages, such as Python, where such distinctions are not readily visible at the syntactic level. We therefore opted for the following distinction between indirection and references: while both pertain to aliasing, indirection is used to specifically refer to lower-level aliasing through pointers, whereas references are used to specifically refer to higher-level aliasing mechanism not involving explicit pointers. Other delineations may be useful in contexts with different curricula.

2.2 Constructive Alignment

Constructive Alignment (CA) exists in the intersection of constructivist learning theory and instructional design (Biggs, 1996), which has been used to develop syllabi in university level education (Brabrand and Dahl, 2007). This approach is based on the view that students construct knowledge through learning activities and that teaching and assessment methods should align with intended learning outcomes (Biggs and Tang, 2015). Further, the focus lies on what students should be able to accomplish: therefore, the desired outcomes should be expressed using verb(s), which are then included in activities and assessments (Biggs and Tang, 2015). Biggs (1996) also states that CA can be used at all levels to guide decision makers when designing higher education. While originally intended and presented as a student-centered approach to teaching, it is often misused as a quality assurance tool (Loughlin et al., 2021).

2.3 Misconceptions and Problematic Knowledge

Notable work has been done to explain why students struggle when learning programming. Misconceptions have been part of that discussion since 1983 (Bayman and Mayer, 1983) and many have been identified since. A misconception describes knowledge that is incomplete and liable to change at any time (Sorva, 2013). Qian and Lehman (2017) present three types of knowledge: syntactic, conceptual, and strategic. Misconceptions describe problematic conceptual knowledge.

Most closely related to the subtle concepts investigated in this paper are misconceptions related to students' mental models. Mental models are established early, and flawed ones lead to issues in tracing code and understanding the execution of code (Mirza et al., 2019). These are also the types of issues that Haglund et al. (2021) and Strömbäck et al. (2023) found students to have with subtle concepts.

Qian and Lehman (2017) summarized that many misconceptions relate to students' existing knowledge, such as strategies, syntax, and mental models, and that it is therefore important to pay attention to their existing knowledge. The authors identified two additional challenges, namely instructors' own pedagogical content knowledge and their understanding of their students' challenges. Lu and Krishnamurthi (2024) investigated misconcep-

tions, several of which relate to these subtle concepts, and the authors noted that many arose even without including advanced programming features.

Research into misconceptions explore conceptual knowledge, but previous research that establish students struggles with subtle concepts (Haglund et al., 2021; Strömbäcket al., 2023) has yet to determine whether those issues stem only from conceptual knowledge. Therefore, we find it prudent to add nuances that can be found in other perspectives on students' knowledge. In this case, we will include nuances from fragile and problematic knowledge.

Perkins and Martin (1985) define four categories of fragile knowledge: *partial knowledge, inert knowledge, misplaced knowledge*, and *conglomerated knowledge*. Students with *partial knowledge* of some concept may, for instance, know how to use a print statement to create a blank line but be unable to use that same statement to insert a carriage return. *Inert knowledge* is knowledge that the student has but is unable to recall or utilize when they need it. This could be knowledge about a concept that a student can provide if asked, but fail to recall on their own when needed. In these cases, a hint can lead the student to a solution. *Misplaced knowledge* is when knowledge related to some concept infiltrates other situations and causes problems. Typically, this may be recently acquired knowledge that is applied improperly, as a result of the student assuming that the last thing they studied somehow must be applicable. *Conglomerated knowledge* is knowledge that can be seen when a student expresses their intent but fails to adhere to the strict rules defined in the programming language. In these cases, the student explicitly seeks to encode information in the program, but does so in a way that does not follow the syntactic or semantic rules.

Perkins (1999) discusses problematic knowledge from the perspective of constructivism, and presents several categories. These categories are *inert knowledge* (already defined by Perkins and Martin (1985)), r*itual knowledge, conceptually difficult knowledge*, and *foreign knowledge*. *Ritual knowledge* is characterized by lack of meaning and routine use. This knowledge lacks meaning and can occur when isolated examples are used during teaching, an example of such knowledge is the division of fraction rule (Hill, 2010). *Conceptually difficult knowledge* is knowledge that clashes with our everyday impressions, common in science and mathematics (Perkins, 1999). For example, the common belief that heavy objects fall faster than light objects (Hill, 2010). *Foreign knowledge* is troublesome knowledge because it does not align with the student's own perspective, though unlike conceptually difficult knowledge, this perspective is not shaped by everyday impressions (Perkins, 1999). For example, students may have difficulty understanding how people behaved during a war, since that situation is foreign to them.

2.4 Addressing Misconceptions

Qian and Lehman (2017) calls for research to not only document misconceptions, but to also strive towards understanding the process of knowledge acquisition. Further, they indicate that the instructor is an important part in addressing misconceptions, noting that the development of their pedagogical content knowledge and their understanding of students'

misconceptions is important for achievement of desired outcomes.

Some CS concepts need to be made explicit to students over time (Koppelman and van Dijk, 2010; Hazzan, 2008; Armoni, 2013). Abstraction is one of these, a concept for which subtle concepts are considered prerequisites (Haglund et al., 2021). Guidelines presentedas part of the PGK (Perrenet, Groote, Kaasenbrood) framework for teaching algorithmic abstraction include that educators be precise, patient and explicitly allow students to engage with the concept (Armoni, 2013).

Abstraction needs to be taught early and learning it is a gradual process (Koppelman and van Dijk, 2010; Hazzan, 2008), one that does not necessarily benefit from having specific courses dedicated to it but rather from being brought up when appropriate in courses (Hazzan, 2008).

Kaczmarczyk et al. (2010) notes that no panacea exists to address misconceptions, but that each instructor has to use their own creativity to address the issues. This is a context where Perkins (1999) insights can be useful. When addressing problematic knowledge, Perkins (1999) suggests that students need to take an active role in their learning to solve these issues. If the knowledge is inert, the student should tackle problems where they must apply it. If the knowledge is ritualized the student should be put in a situation where they need not only use that knowledge but try to form a deeper understanding of it, by for instance, discussing it. If the knowledge is conceptually difficult, the student can be encouraged to tackle qualitative problems relating to that knowledge rather than quantitative ones. If the knowledge is foreign, students can explore other perspectives relating to that knowledge. Perkins (1999) recognize that knowledge can be troublesome in other ways and that other solutions can exist; nevertheless, having students actively work with organizing knowledge is desirable.

Hewett (2005) found that when people work with solving problems, it is important to minimize the strain on working memory, while active engagement promotes retention. Notional machines are often used to reduce cognitive load for learners (Fincher et al., 2020). The function of notional machines is to draw focus to something, and can thus be used to help students focus on cognitively demanding concepts such as references or aspects that are otherwise invisible.

Du Boulay et al. (1999) found that when learning programming, two important characteristics of programming languages are visibility and simplicity. Simplicity can be achieved by limiting the systems' scope and visibility by different techniques, such as turtle programming (Du Boulay et al., 1999).

3. Method

In this section, we describe the method used in this study. In order, we describe: the CS programs being investigated, selection of courses from those programs, the survey used to collect data, validity and reliability, and the statistical analysis.

This study is of an exploratory nature. We aim to describe teachers' perceptions, rather than measuring effectiveness, within these three CS programs, thus contributing to a richer understanding of how these concepts are approached in practice. We acknowledge that the findings are context-bound and not intended for broad generalization; instead, they offer empirically grounded insights that may inform future research and pedagogical development in similar educational contexts.

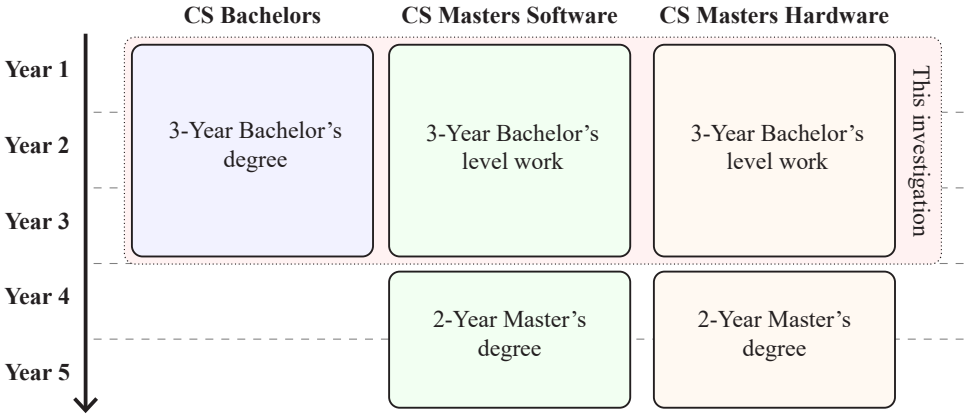

Figure 1. Overview of the time-frame of the three CS programs and which parts are investigated in this study.

3.1 Description of CS Programs

This section briefly describes the three CS programs that were investigated in this paper. While the programs have elective and mandatory courses, only the mandatory courses are considered. These programs are the same CS programs that have previously been shown to not significantly improve students' grasp of subtle concepts (Strömbäck et al., 2023). Since that study covered the development over the first three years, we will also focus on courses during the same time span (Figure 1). While the previous study measured student performance, the current study is oriented around the teachers' perception of students' proficiency with subtle concepts and endeavors to understand students' progression across courses. We here present a brief summary of how the programs are defined in their respective syllabi:

CS Bachelors. This Computer Engineering program* is a three-year, 180 ECTS engineering degree focused on the analysis, design, and implementation of computer-based systems. The curriculum integrates mathematics, computer science, and electronics to cover both hardware and software aspects of digital systems.

Students are expected to acquire competencies in programming, software engineering methods, digital and embedded systems, computer architecture, operating systems, net-

*https://studieinfo.liu.se/program/6IDAT/5735#syllabus

working, and data management. The program includes theoretical coursework, laboratory exercises, and project-based activities that require the modelling, implementation, and evaluation of technical systems.

Students are also expected to develop abilities in problem formulation, system modelling, experimentation, project coordination, and technical communication. After a common introductory phase of three semesters, students select a specialization in either Software or Embedded Systems. The program concludes with an independent project within the field of computer engineering. In this paper, only the software specialization is considered, since it has the most programming-oriented courses and is the most popular specialization among students.

CS Masters Software. This Software Engineering program* is a five-year*, 300-ECTS engineering degree focused on the development and analysis of software-intensive systems. The courses overlap to an extent with the CS Masters Hardware program. Entering this master's program does not require a previous bachelor's degree. The curriculum combines mathematics, computer science, and software engineering, covering areas such as algorithms, programming, distributed and embedded systems, data management, artificial intelligence, and human–computer interaction.

Students are expected to acquire competencies in software engineering methods, system modelling, experimental investigation through implementation, and the evaluation of software systems. They are also expected to develop abilities in problem formulation, project coordination, and technical communication.

During the final two years, students select a specialization profile and take advanced courses, concluding with an independent master's thesis within computer science, computer engineering, or information technology.

CS Masters Hardware. This Computer Engineering program* is a five-year, 300-ECTS engineering degree integrating mathematics, computer science, and electrical engineering. Entering this masters program does not require a previous bachelor's degree. The program covers programming, algorithms, computer architecture, embedded systems, electronics, signal processing, and control.

Students are expected to acquire competencies in mathematical modelling, engineering problem formulation, experimental investigation through implementation, and the analysis of computer-based systems. They are also expected to develop abilities in system thinking, project coordination, group work in defined roles, and technical communication.

During the final two years, students choose a specialization profile and complete advanced coursework within an approved subject area. The program concludes with an independent master's thesis.

---

*https://studieinfo.liu.se/program/6CMJU/6293#syllabus

*The CS Masters Software and Hardware are combined bachelor's and master's programs. They are designed to be taken by students in five consecutive years. With the first three years corresponding to a bachelor's program and the last two years corresponding to a two-year master's program.

*https://studieinfo.liu.se/program/6CDDD/5723#syllabus

3.2 Course Selection

The following criterion was used to select courses that were of interest to this study: For students in this course, is programming a central part of achieving course objectives? Any course that might meet this criterion based on its description was included.

Courses were selected in two stages. First, the first author selected candidate courses based on course plans for the mandatory courses in the programs. Courses that obviously did not involve programming as a central part of achieving course objectives were excluded. In this process of exclusion, we erred on the side of caution, rather including than excluding courses. For each course that conceivably was of interest to the study, the corresponding responsible teacher was contacted and asked to participate in the study. Before responding to the questions used in the analysis, the teachers were presented with the inclusion criteria and asked to determine whether their course met them. This way, we could ensure that only relevant courses were included in the study.

The review of course plans yielded 29 courses that were of interest, overseen by 23 teachers. Each responsible teacher was invited to participate in the study by filling out the survey for the courses they were responsible for.

The curricula for the programs investigated, as well as syllabi for individual courses in those curricula, are available online. Links for the courses are provided in the appendix section Section A. The description of the courses, presented in Section 4, are brief summaries of the syllabi. All credits are European Credit Transfer and Accumulation System (ECTS) credits*. 60 ECTS credits are equivalent to a full year of study. The studied bachelor's program is 180 ECTS credits and the master's programs are 300 ECTS credits.

3.3 Data Collection

We developed a survey to gain insight into how the courses of CS programs support the learning of subtle concepts. The survey was designed in an iterative manner and is the product of discussions among the authors. The survey focused on five questions, collecting data for each subtle concept on a scale from 1 to 5. In addition, the survey included metadata questions, such as course name, open-ended questions letting respondents leave text comments, as well as the option to end the survey early if the respondent did not think their respective course met the inclusion criterion. The full survey is included in Section B.

Each participant was presented with a set of questions in the form of a digital survey. After being asked for the name of their course, the instructor was asked if programming was a central part of achieving the course objectives. If the respondent answered that their course did not meet the inclusion criteria, the survey would give them the chance to comment before finishing. Thus, respondents that did not consider programming to be a central part of achieving the learning objectives in their course would only answer three questions in total. The survey was designed in this way to reduce the time spent by instructors whose

*https://education.ec.europa.eu/education-levels/higher-education/inclusive-and-connected-higher-education/european-credit-transfer-and-accumulation-system

courses were not of interest for the study. Other respondents were presented with 13 questions. The five questions analyzed in this study are listed below. Each question was asked for all four subtle concepts respectively:

Q Prerequisites "When first starting to take this course, students have a good understanding of [one of the subtle concepts]". Scale was from "Strongly Agree" (5) to "Strongly Disagree" (1).

Q Taught "The following concept is taught in the course [one of the subtle concepts]". Scale was from "Explicitly" (5) to "Implicitly" (3) to "Not at all" (1)

Q Assignments "When working with assignments students need to have a good understanding of this concept in order to solve the assignment: [one of the subtle concepts]". Scale was from "Strongly Agree" (5) to "Strongly Disagree" (1)

Q Graded "The following concept is graded by course staff: [one of the subtle concepts]". Scale was from "Explicitly" (5) to "Implicitly" (3) to "Not at all" (1).

Q Requirements "Considering how the course is designed and structured, how good does a student's understanding of these four programming concepts have to be in order to achieve a passing grade in the course?" followed by each subtle concept. Scale was from "Very Good" (5) to "Very Poor" (1).

Invitations were sent via e-mail and included a link to the survey, a list of courses held by the instructor that we deemed might be relevant to this study, information about the purpose of the study being conducted, information about anonymity, and the voluntary nature of participation. In cases where a teacher taught many similar courses, they were offered the option of filling in the survey once for multiple courses instead if they felt that there were no differences in the courses' relation to these concepts.

The invitation to participate was sent in the month of April 2024. Multiple reminders were sent out to teachers that had not responded. The last teacher to respond to the survey did so in June of the same year. Though, several more reminders were sent out to teachers that had not yet responded. Of the 23 invited teachers, 20 chose to participate, covering 23 out of 29 courses of interest. Respondents reported that in 5 of those courses, programming was not a central part in achieving objectives, leaving us with 18 courses. One respondent filled out the survey once for multiple courses. The data collected were analyzed statistically using nonparametric Welch t-tests. We also contextualize the answers for the courses with previously established guidelines for teaching subtle concepts.

3.4 Validity and Reliability

Since the survey explores teachers' perceptions, it is not necessarily useful to compare the absolute values between different courses. Most courses for which the data were collected have different teachers. Each of these teachers likely have a different idea of what a response to a question indicates. As such, it is not productive to look at a pair of courses with different teachers and compare their scores directly. Therefore, we should not con-

clude that there is some sort of issue with how a course is assessed if Q Requirements scores highly in one course and a subsequent course scores low on Q Prerequisite. What the teacher in the first course considers to be a good understanding of one of the four subtle concepts is likely governed by how that concept relates to the main concepts covered in that course and their desired learning outcomes for those concepts.

However, this problem does not exist within a course. If the data indicate that a teacher thinks that students' prerequisites for a given course are poor, while they would need a good grasp of the subtle concepts to pass the course, that is the perception of one instructor and in the context of the learning objectives of a single course. This is sound for the validity of the method, since it is the same person's perception of what constitutes a good grasp of the concepts. Since the number of courses is fairly small, the statistical analysis will only be able to find significance in fairly strong trends.

3.4.1 Institutional Context and Definitions

A limitation of the survey instrument concerns the degree to which the questions were operationally defined. While the items were designed to capture instructors' perceptions of students' proficiency with the selected subtle concepts, several of the constructs were not defined with sufficient precision to ensure consistent interpretation across respondents. Consequently, individual instructors may have interpreted certain terms or response categories differently, depending on their own teaching context and experience. However, since all respondents were employed within the same institution and shared a common curricular framework, it is reasonable to assume that there existed a shared understanding of key concepts and program structures. This institutional homogeneity may have mitigated some of the variation in interpretation, although it also limits the external validity of the findings to other institutional contexts. That is, while it is likely that instructors at this university has a common interpretation of what constitutes explicitly grading a concept, that is less likely outside the institution. When deploying a similar survey, especially to other institutions with other curricular frameworks, it would be valuable to further define the questions.

3.5 Statistical Analysis

The statistical tests performed were Welch t-tests*. Effect size is calculated with Cohen d and confidence interval is calculated using the SciPy library*. Although the survey data were collected on an ordinal Likert-type scale, Welch's t-test was employed, given its established practice of treating Likert data as approximately interval (Sullivan and Jr, 2013). Norman (2010) found that Likert data could be analyzed with the parametric statistical tests even with smaller sample sizes than in this investigation. Further, he explains that these tests can be used without fear of drawing the wrong conclusions even when dealing with non-normal distributions and unequal variances, however, it is still important to ac-

*Tests were performed using Python version 3.10 with the SciPy module version 1.13.0.
*Test performed using Python version 3.10 with SciPy module version 1.13.0.

knowledge that the sample size places limitations on the external validity of the findings. Data from respondents who had answered that programming did not have a central role in achieving course objectives were excluded. The course Project for CS Bachelors was excluded since the teacher stated that they had no insight into students' proficiency with the subtle concepts, as the course focuses on project management strategies. For completeness, data collected for this course, called Project, are still included in Figure 4.

The tests performed were:

3.5.1 Differences between concepts:

In this test, we compare if the subtle concepts are distinct from each other. The null hypotheses is that the subtle concepts are the same. There are four subtle concepts requiring six tests to cover all combinations of the subtle concepts. Since data was collected for each subtle concept for each question, we make this comparison separately for each question. Since there are five questions, we therefore need 30 tests in total to cover all combinations.

Therefore we consider a test result significant if $p < 0.05/30 \approx 0.0017$.

OOP C++

	P	**T**	**A**	**G**	**R**
I	3	5	5	5	3
R	2	3	4	4	2
S	2	5	4	3	4
P	4	3	5	3	5

Figure 2. Example of how the responses to the survey is visualized as a table in this study. Each table represents one respondent's answers to the survey. Each column corresponds to one of the five questions in the survey. Each row corresponds to one of the subtle concepts. Thus, each cell corresponds to a respondent's perception of a specific subtle concept for a specific question. If we wanted to know if this respondent thought that indirection is taught explicitly in this course: we would look at the first row and second column and find that indirection is taught explicitly in this course.

The mean of a single cell (Figure 2) is calculated from all 17 responses. The mean of each cell is compared with each other cell in that same column:

Intuitively, all answers were structured into 17 tables like the one in Figure 2. Then, for each column (i.e., each question), we tested for differences between each of the rows in that column (i.e., each subtle concept for that particular question) using data from all 17 tables.

The 30 tests can also be formulated as a comparison between sequences A[*j, k*] below:

i = course

j = subtle concept

k = question

r_{ijk} = response for how course i utilizes subtle concept j for question k

$A[j, k] = \langle r_{ijk}$ for all $i \rangle$

$A[j, k]$ compared to $A[l, k]$ where $j < l$

3.5.2 Differences between the questions:

We investigate whether there is any significant difference between the questions. For all responses, a mean is calculated for each question (column in Figure 2). The mean for each column is then compared to the mean of each other column. As there are 5 questions on the survey, 10 tests were conducted to cover all combinations. Therefore we consider a test result significant if $p < 0.05/10 = 0.005$.

Intuitively, in terms of the table in Figure 2, we treat all rows of all 17 tables as one, and test whether there are any differences between the columns (i.e., between the different questions).

Again, this can be formulated as a comparison between sequences $B[k]$ as below. The variables $i, j, k,$ and r_{ijk} are the same as for the previous test.

$B[k] = \langle r_{ijk}$ for all $i, j \rangle$

$B[k]$ compared with $B[l]$, where $k < l$

4. Results

In this section, the results from the statistical analysis are presented, then the context of the first three years of each CS program.

4.1 Statistical Analysis

4.1.1 Differences Between Concepts:

Across the 30 Welch t-tests conducted, none yielded statistically significant results with the adjusted threshold of $p < 0.0017$. Thus, we found no evidence to reject the null hypotheses that instructors' perceptions of the concepts are the same. In other words, the statistical analyses did not indicate systematic differences in how instructors perceived, for example, the extent to which students' understanding of indirection, references, scope, and parameter passing was assessed.

4.1.2 Differences Between the Questions

When subtle concepts were combined, most tests showed that the differences were significant, as can be seen in Table 1. Students' prerequisite knowledge (Q Prerequisites) is significantly smaller than 1) how explicitly the concepts are taught (Q Taught), 2) the grasp needed of subtle concepts when solving assignments (Q Assignments), 3) how explicitly the concepts are graded (Q Graded), and 4) the grasp needed to achieve a passing grade (Q Requirements).

The students' grasp of the concepts when working with assignments (Q Assignments) is significantly greater than how explicitly the concepts are graded (Q Graded) and the grasp needed to achieve a passing grade (Q Requirements).

Table 1. Table showing the results of Welch t-tests between the questions on the survey. The p-value column reports the p-value rounded to 3 decimals. The t-test column, effect size, and confidence shows the value rounded down in absolute terms. Highlighted rows are significant ($p < 0.005$).

Question	Question	$p <$	$t <$	Effect	Confidence
Prerequisites	Explicitly Taught	0.001	-4.248	-0.729	[−1.337, −0.487]
Prerequisites	Assignments Need	0.001	-7.536	-1.293	[−1.782, −1.041]
Prerequisites	Explicitly Graded	0.001	-3.965	-0.582	[−1.141, −0.301]
Prerequisites	Assessment Requires	0.001	-3.937	-0.675	[−1.060, −0.351]
Explicitly Taught	Assignments Need	0.026	-2.252	-0.386	[−0.939, −0.061]
Explicitly Taught	Explicitly Graded	0.434	0.785	0.135	[−0.290, 0.672]
Explicitly Taught	Assessment Requires	0.341	0.956	0.164	[−0.220, 0.632]
Assignments Need	Explicitly Graded	0.003	3.147	0.540	[0.257, 1.126]
Assignments Need	Assessment Requires	0.001	3.754	0.644	[0.334, 1.078]
Explicitly Graded	Assessment Requires	0.946	-0.69	0.012	[−0.407, 0.436]

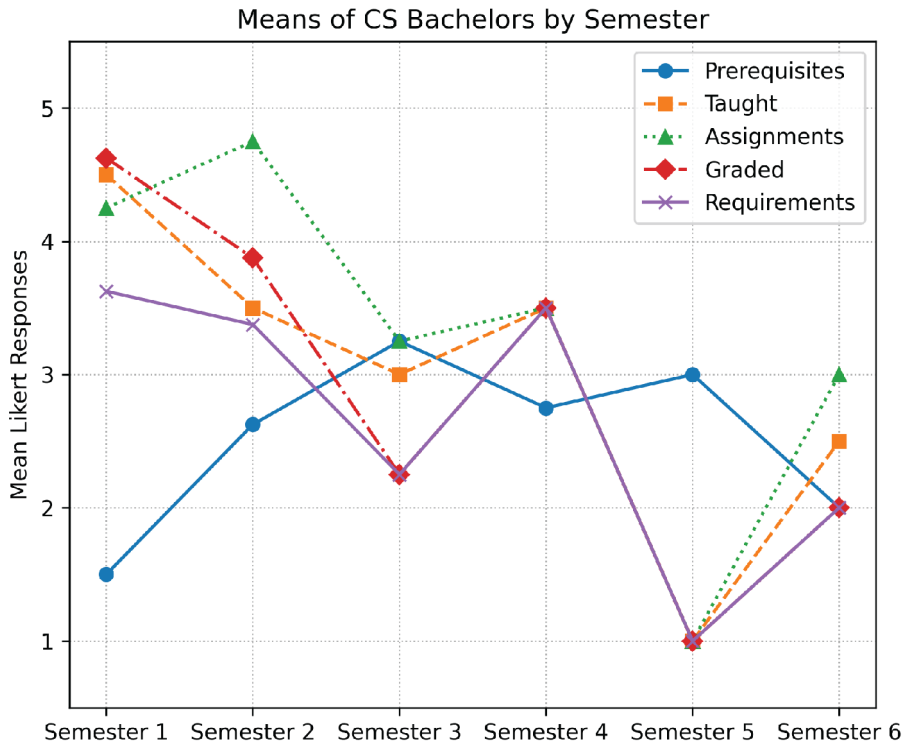

Figure 3. Overview of the mean Likert responses for the bachelor's program by semester. For each semester, the mean response was calculated across all relevant courses. For example, the value for Taught in Semester 1 represents the mean of the teachers' responses for the courses Intro Ada/C++ and Intro Assembler across all concepts.

4.2 CS Bachelors

In this section, the results from the courses in the CS Bachelors program are presented. A graph showing relevant courses and the data collected for them (Figure 4), a brief description of each course in (Table 2), and a summary of notable things in the data with a graphical illustration of the means of the responses by semester in Figure 3.

In the CS Bachelors program, all concepts are taught explicitly in the introductory course, with the respondent reporting 5s for all subtle concepts in Intro Ada/C++ (Figure 4). The concepts are also taught at least somewhat explicitly in other courses during the first year, with all concepts scoring at least a 4 in one of the three other courses during the first year (Figure 4). An illustration of this can also be found in Figure 3.

Subtle concepts are explicitly graded during the first year (Figure 3), with each concept scoring at least a 4 more than once except scope that is only explicitly graded in the Intro Ada/C++ course. The concepts are less explicitly graded after the first year. No concepts

scores greater than a 3, except for indirection and references that score a 4 in Concurrent Programming (C).

Most teachers find that students' prerequisite grasp of the concepts is not good. Concepts only score greater than a 3 in four courses: parameters in OOP C++, scope in C++ Project, references in Algorithms and DS (C++), and scope in Concurrent Programming (C). The concept(s) scoring high on prerequisites is not consistent between courses either (e.g., parameter passing scoring high and scope scoring low in OOP C++, but the opposite being true for C++ Project immediately afterward). This program has fewer changes of languages between courses than both CS Masters, having four courses back-to-back in C++, which they then follow up with a similar language in C and then a self or customer selected language in the project course. To achieve passing grades in the courses, educators think students need to have a good understanding of each of the concepts at the end of the first year.

Two courses of note were excluded from the graph (Figure 4): Distributed Systems (8 ECTS credits) and Thesis Work (15 ECTS credits). We received no response to the survey from the teacher of Distributed Systems, the teacher did however verbally confirm that programming was not a central part of achieving the course objectives. The bachelor program students conduct Thesis Work during their 6th semester. While this work can, and often does, include programming tasks, it is not always the case. The assessment of the thesis work is focused on the report, even if software is often produced as part of that work.

Table 2. Short description of the CS Bachelors courses in Figure 4. Credits are ECTS credits. While courses that are similarly named to other courses do overlap to different extents, they are not the same course.

Course	Credits	Prerequisites	Outcomes	Language
Intro Ada/C++	6	None	Create programming solutions to problems with an imperative approach, create abstractions using functions and simple datastructures, know about and be able to use two programmings languages imperatively.	Ada and C++
Intro Assembler	4	None	Program a processor using assembler programming and to use a processor in an integrated system.	Assembler
OOP C++	4	Imperative C++	Use designprinciples, methods, and techniques in object orientet programming to create programs that solves given problems	C++
C++ Project	10	OOP in C++	Create solutions to problems with an object oriented approach, to solve data manipulation problems with careful selections of parts from the standard template library, create your own class and function templates.	C++
Algorithms and DS	6	C++	To explain and use common data structures and algorithms.	C++
Concurrent Programming	4	OS, DS, Prog	To implement basic system calls in the example operating system Pintos, to explain synchronization terms, and to analyze program code to identify critical sections and active waiting and to be able to use relevant mechanisms to adequately correct faulty code and to make it thread safe.	C
Project	8	Prog	To develop a graphical interactive system together with other programmers and an external customer, to develop software using the tenets, methods and tools of agile systems development, to reson about ethical aspect when developing technical systems. The responsible teacher stated in the survey, that while programming is an important part of achieving the course objectives, the course staff is not really involved with the programming aspects of the course, but instead focus on the project development techniques.	Any
Web Programming	6	OO Language	To use and explain techniques for creating interctive web content and were appropriate create connections to databases, create a single page application, explain the principles of web services and using some simple technigques for creating and calling web services.	Pyhon, Javascript

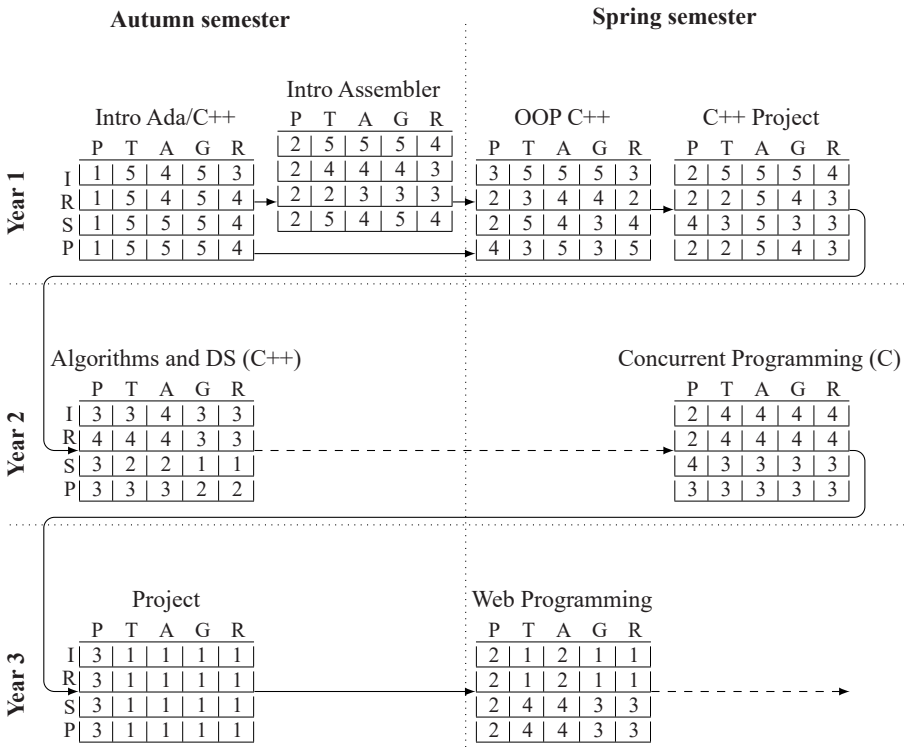

Figure 4. Overview of the CS bachelor program. Each table represents a course where programming is central. Each column represents one question on the survey for which 4 answers are given on a scale from 1 to 5. Each of the four answers corresponds to one subtle concept: Indirection, Reference, Scope, and Parameter Passing. A filled arrow from one course to the next indicates that the course continues until the course which the arrow terminates. A dashed arrow indicates that the course ends at the beginning of the dashed arrow and that no new programming centric course starts until where the arrow terminates.

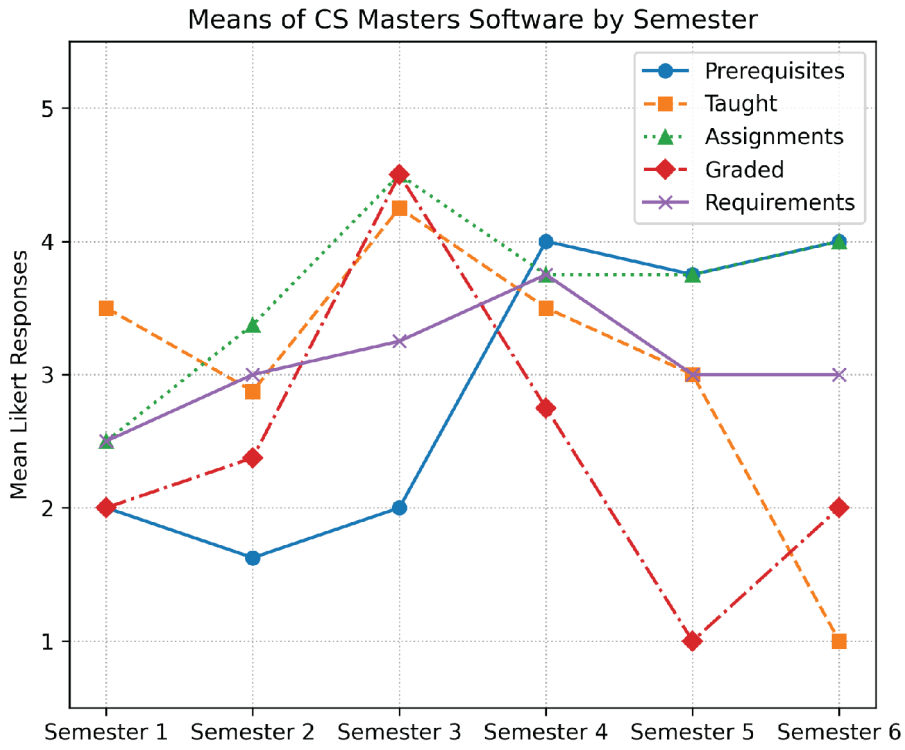

Figure 5. Overview of the means of Likert responses for the CS Masters Software program by semester. The means of the responses for each course have been calculated by semester and plotted. I.e. the point for Taught in Semester 2 is the mean of what teachers reported for Taught in the course: Assembler Intro and App Project (Python). Courses for which no data was collected have been excluded (such as Python Part 2).

4.3 CS Master Software

In this section, the results from the courses in the CS Bachelors program are presented. A graph showing relevant courses and the data collected for them (Figure 6), a brief description of each course in the graph, and a summary of notable things in the graph along with an illustration of the means of teachers responses by semester (Figure 5).

Though data is missing for two courses in the first year (from the same teacher), all the subtle concepts are taught at least somewhat explicitly during the first year in the CS Master Software program (Figure 6). All concepts, except references, score a 4 or greater in Python Part 1 and references and indirection scores a 4 or greater in App Project (Python). All concepts, except for scope, are taught explicitly in the second year. Scoring 5s in Algorithms and DS (C++) and Concurrent Programming (C). No respondent reported a higher score than 3 for explicitly teaching the concepts in the third year.

Teachers experience that students have lacking prerequisite knowledge of subtle concepts

until the 4th semester starts (Figure 5). During the first three semesters, no subtle concepts scores higher than 3 on prerequisites, except for scope in Algorithms and DS (C++). After that, they are experienced as good, or at least not bad. In the third to sixth semester, no concept scores lower than 4 on prerequisites, except for scope in AI (Python/Java) that scored a 3.

In contrast to the CS Bachelors, the CS Masters Software program does switch programming languages frequently (Figure 6). There is no gap between courses where these concepts are touched upon that is longer than half a semester. While most (5/7) respondents report that students need a firmer grasp of some of the concepts than they have as prerequisites to achieve a passing grade, how explicitly the concepts are graded varies. Explicit grading of all the concepts stops after the third semester. The last concept scoring a 4 on grading being indirection in Concurrent Programming (C).

The course Distributed Systems (11 ECTS credits) was excluded from the graph. We received no response to the survey from the teacher of Distributed Systems, the teacher did however verbally confirm that programming was not a central part of achieving the course objectives.

Table 3. Short description of the CS Master Software's courses in Figure 6. Credits are ECTS credits. Prerequisites and outcomes are abbreviated and only aspects related to the concepts of interest are included. While courses that are similarly named to other courses do overlap to different extents, they are not the same course.

Course	Credits	Prerequisites	Outcomes	Language
Python Part 1	6	None	Implement and design simple algorithms, solve programming related problems systematically, work interactively while implementing, testing, and debugging programs, constructing programs in Python.	Python
Python Part 2	5	Python part 1	Explain basic computer science concepts that relate to programming and programming languages, to design and implement recursive and iterative algortihms, and to build abstractions with varying degrees of support from the programming language.	Python
Assembler Intro	6	Some Programming	Program a computer in Assembler with IO and interuptions.	Assembler
App Project	11	Imperative, functional, OOP	To design, implement, and evaluate basic web based software services, to design, implement, and evaluate usefull and well testested mobile applications that contain social elements, to reflect on and apply relevant methodology and practices for software development projects.	Java, Python
Java	6	Python part 1, 2	Explain and apply fundamental concepts of object oriented programming, to apply different aspects of datatypes in object oriented and non object oriented languages, and to implement object oriented programs in Java.	Java
Algorithms and DS	11	functional, imperative, and OOP	Implement and understand Algorithms and DS	C++
Concurrent Programming	6	C/C++, Hardware, DS	Understanding and programming operating systems and to deal with issues of synchronization and parallellism	C
AI	5	functional, imperative, or OOP. Algorithms, DS.	explain and discuss terms withing artificial intelligence and to use well known techniques withing artificial intelligence	Java/ Python
Bachelor Project	15	functional, imperative, OOP	Systematically integrate the knowledge they have gained so far, specifically withing programming and computer science, including methods and to create, analyze and/or evaluate techincal solutions.	Any

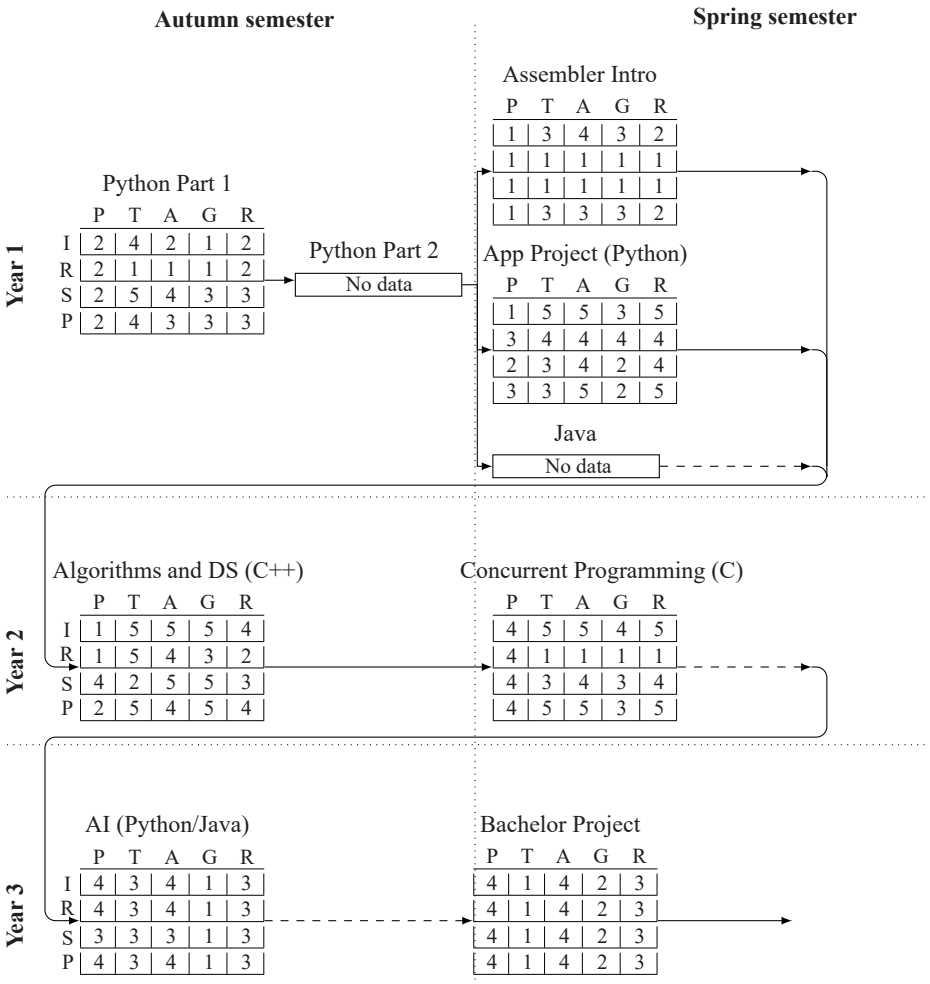

Figure 6. Overview of the CS Masters Software program. The legend is explained in Figure 4 is consistent with this figure.

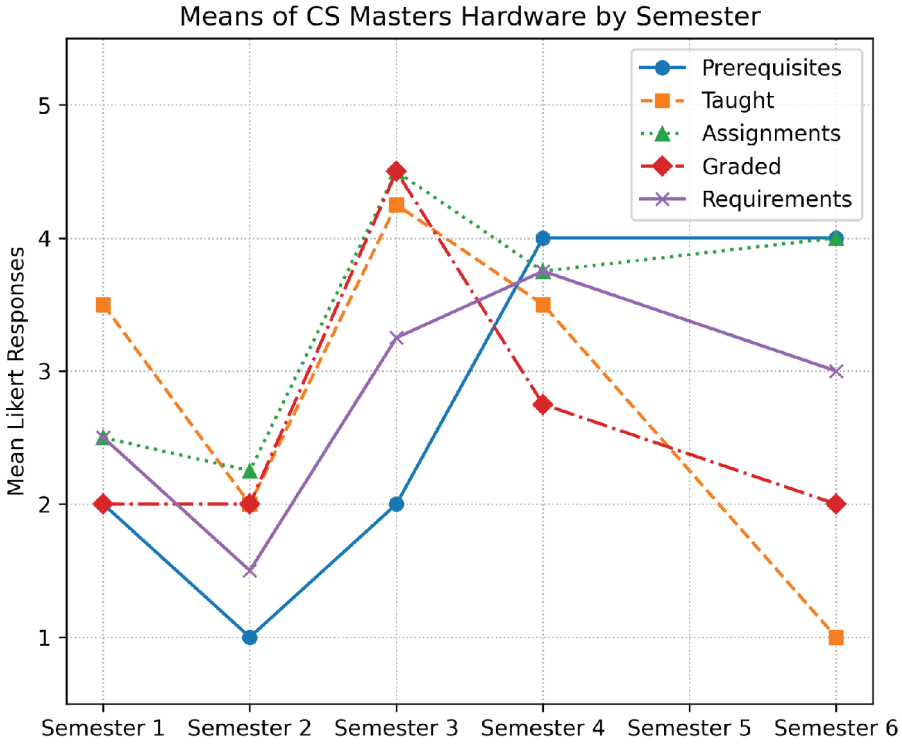

Figure 7. Overview of the Likert responses for the CS Masters Hardware program by semester. Courses for which no data was collected have been excluded (Assembler 3). Unlike the other programs, we had no case where data existed for more than one course during a semester. Thus, this representation does not show the mean value of several courses, but rather just illustrates the means of each column found in the graph (Figure 8). Note the missing markers in Semester 5 since we do not have any data for this program during that semester.

4.4 CS Master Hardware

In this section, the results from the courses in the CS Bachelors program are presented. A graph showing relevant courses and the data collected for them (Figure 8), a brief description of each course in the graph, and a summary of notable things in the graph along with an illustration of the responses by question and semester in (Figure 7).

Students in the CS Master Hardware program are at least somewhat explicitly taught all concepts, except for references during the first year (Figure 8). Prerequisites are poor, with the majority scoring 1s or 2s, until the 4th semester (Figure 7). After the 4th semester, the data indicate that prerequisites are adequate. Though we are missing data for some courses, it is likely that the longest gap between programming centric courses in the curriculum is,

at most, half a semester. Data for four out of the five course show that the requirements for passing the course are higher than the prerequisites for two or more of the subtle concepts, which is also illustrated in Figure 7, where we see the line for Requirements is consistently below the line for Prerequisites until the fourth semester. The concepts are explicitly graded for the last time during the third semester. The Distributed Systems course was excluded from this graph, this is the same course described in Section 4.3

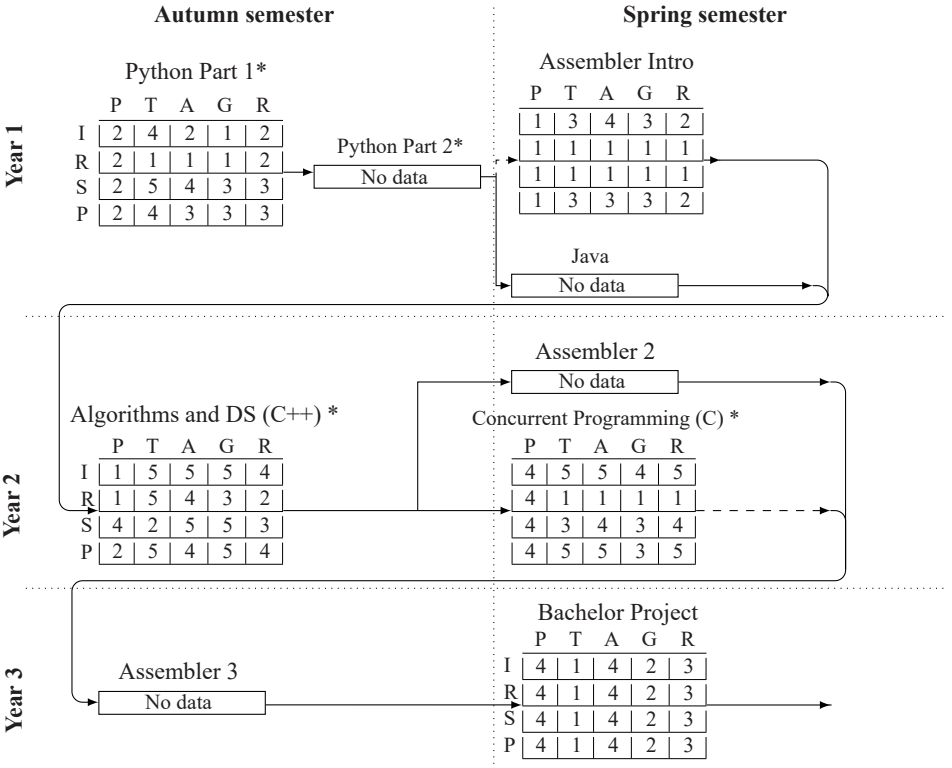

Figure 8. Overview of the CS Master Hardware program. The legend explained in Figure 4 is consistent with this figure. Courses marked with an asterisk (*) are duplicated from Figure 6. Assembler 3 lists C++ and assembler programming in the syllabus (Figure 8)

Table 4. Short description of the CS Master Hardware's courses in Figure 8. Credits are ECTS credits. Prerequisites and outcomes are abbreviated and only aspects related to the concepts of interest are included. Courses present in Figure 8 and not present in this table are identic to the course listed in Table 3. While courses that are similarly named to other courses do overlap to different extents, they are not the same course.

Course	Credits	Prerequisites	Outcomes	Language
Assembler Intro	4	Some Programming	Program a computer in Assembler and interuptions.	Assembler
App Project	11	Imperative, functional, OOP	To design, implement, and evaluate basic web based software services, to design, implement, and evaluate usefull and well testested mobile applications that contain social elements, to reflect on and apply relevant methodology and practices for software development projects.	Java
Java	7	Python part 1, 2	Explain and apply fundamental concepts of object oriented programming, to apply different aspects of datatypes in object oriented and non object oriented languages, to implement object oriented programs in Java, and to present results and conclusions in a linguisticly correct written report.	Java
Assembler 2	8	Assembler, hardware, architecture	Use assembler- and microprogramming to gain knowledge of how computers work at the lowest level	Assembler
Assembler 3	8	Previous assembler, hardware, architecture, concurrent programming, OS, C/C++	None of the intended learning outcomes relate directly to programming, though many could be argued to be adjacent	Assembler, C/C++
Bachelor Project	15	functional, imperative, OOP	Systematically integrate the knowledge they have gained so far, specifically withing programming and computer science, including methods and to create, analyze and/or evaluate techincal solutions.	Any

5. Discussion

In this section, we will discuss key aspects emerging from the results.

5.1 Teaching the Concepts Consistently and Explicitly

Across the three CS programs, the patterns in instructors' reported teaching practices suggest that students are exposed to the subtle concepts to varying degrees throughout the first three years. During the first year, students typically encounter at least somewhat explicit instruction in most of the concepts (Figs. 4, 6 and 8). The exception appears in the CS Masters Hardware program, where explicit instruction in references was not reported (Figure 8). Given that this program includes first year courses with learning objectives involving recursion and object-oriented programming that we failed to collect data for (Tables 3 and 4), it is plausible, but not verifiable, that the concept is taught but not captured by the survey responses. During the second year, most concepts are taught more explicitly (levels 4–5), while instruction in the third year appears less explicit overall. This shift may reflect the structure of the curriculum, in which students engage more with project-based courses during later semesters. For the third-year non-project courses for which data were available (Web Programming, AI), instructors still reported at least implicit or somewhat explicit inclusion of several concepts (Tables 2 and 3).

Previous work has suggested that subtle concepts benefit from being taught consistently and explicitly (Fisler et al., 2017; Reimer et al., 2018; Strömbäck et al., 2023). Our data indicate that students who do consistently encounter these concepts still do not significantly improve their grasp of subtle concepts. While this study cannot establish why such improvement is lacking, it raises the possibility, worth considering in future work, that merely encountering the concepts in coursework may not, on its own, support deeper understanding. Rather, students need to explicitly engage with the concepts. The explicitness does decline over time, with bachelors being taught explicitly for the last time in the first year and both masters in the second year. Previous research has also found that TAs sometimes consider subtle concepts as something students learned in a previous course and that they therefore do not need to teach it (Haglund et al., 2024). This is potentially in conflict with the perceptions teachers have in this study.

The instructors in this study generally perceived students' grasp of the subtle concepts as good by the end of the third semester across both master's programs. This perception stands in contrast to earlier work (Strömbäck et al., 2023), which reported no significant improvement in students' performance on these concepts over the same period. Although we cannot account for this discrepancy using the approach of this study, it does support previous findings on the importance of raising awareness among instructors about students misconceptions (Qian and Lehman, 2017). More detailed investigations of how instructors form these perceptions, and how they monitor students' conceptual development over time, could be a useful direction for future research.

5.2 Teachers Perceive that Students Improve

The knowledge that teachers perceive students to have at the beginning of courses is statistically lower than what is required in the course, both when students work with assignments and what they need to achieve a passing grade. This is shown in the figures (Figs. 4, 6 and 8) and the statistical tests (Table 1).

This aligns with previous recommendations (Strömbäck et al., 2023; Qian and Lehman, 2017; Piaget, 2003; Fisler et al., 2017). At the same time, previous work shows no corresponding improvement in measured student performance for these programs (Strömbäck et al., 2023). This highlights a potential area where instructors' perceptions of progression and students' demonstrated progression may not align. Furthermore, several courses appear to require a stronger grasp of subtle concepts for assignments than is reflected in how explicitly those concepts are graded in the same courses (Table 1).

Further, the grasp of subtle concepts required by students when engaging with assignments is perceived as greater than what is required by assessments and how explicitly the concepts are graded (Table 1). This points to a potential issue with the alignment of activities and assessments (Biggs and Tang, 2015) within courses, although it cannot be verified based on the scope of this study.

5.3 Extent of Grading

Instructors reported that students need a significantly better grasp of the subtle concepts than is assessed through explicit grading (Table 1). Previous research emphasizes the importance of making these concepts visible in both instruction and assessment (Fisler et al., 2017; Strömbäck et al., 2023; Perkins, 1999). Teachers perceived that such explicit assessment is limited, particularly outside of courses that serve to introduce particular programming languages to students, such as the ADA/C++ course.

Whether students find ways to complete assignments without fully mastering these concepts cannot be determined from the current dataset. However, the observed mismatch between required understanding and explicit grading raises questions that may help illuminate the lack of improvement observed previously in these programs (Strömbäck et al., 2023). Future work examining how assessment practices relate to conceptual development could offer valuable insights.

5.4 Swapping Languages

The CS Bachelors, CS Masters Hardware, and CS Masters Software are exposed to a different number of programming languages in a different order. A previously established challenge with these types of concepts are that they do not transfer automatically between languages (Fisler et al., 2017). The CS Bachelors have the most homogeneous set of languages. After an introduction in Ada they consistently work with C and C++, with a dash of assembler, for most of their education (Figure 4), languages where the subtle concepts

are made explicit to some extent by the syntactic nature of the language. The mastersstudents have a different approach; starting out in Python before tackling Java and Assembler in parallel with more Python, then switching to C++ and C followed by more Python/Java (Figs. 6 and 8). However, the data collected for the masters programs (Figs. 6 and 8) show that students are explicitly taught most of these concepts when making the transition from Python to C++, as recommended by Fisler et al. (2017). Since the bachelor program did not show a superior improvement to the masters programs over the first three years (Strömbäck et al., 2023), even tough they have fewer language switches, we find support for Fisler et al. (2017) recommendations.

5.5 Students Need to Improve

Teachers think that students need a better grasp of at least some of the subtle concepts to achieve a passing grade or to work with assignments in the course compared to what they have at the start of the course during the first two years of the programs. Examples include all the master students' courses during the first two year, where at least half the concepts have a lower score on prerequisites than what is required to work with assignments or what is required to achieve a passing grade in the course (Tables 3 and 4). Except for the Algorithms and DS (C++) course, the same is true for the bachelor students (Table 2). In the third year, only the CS Bachelors have a course where the students do not start the course with a good enough grasp of the concepts.

While TAs working in CS courses sometimes expect students to have a good understanding of subtle concepts from earlier courses (Haglund et al., 2024), teachers rarely agree that students have a good grasp of the concepts at the beginning of courses (Tables 2 to 4). This observation is further supported by survey responses to Q Prerequisites being significantly lower than Q Assignments and Q Requirements (Table 1), meaning that these teachers think students need to a better grasp of subtle concepts in order to engage with the assignments and achieve a passing grade in their courses.

5.6 Notes of Interest for CS Bachelors

Only half of the respondents perceived students to have a good understanding of any of the subtle concepts at the beginning of their courses. OOP C++, C++ Project, Algorithms and DS (C++), and Concurrent Programming (C) were the only courses where respondents answered 4 or higher for one of the subtle concepts in Q Prerequisites in (Figure 4). Students are still required to reach a good understanding of all concepts in the first year, each concept scoring at least one 4 or better in Q Requirements in Figure 4 and all concepts except for references having at least two such scores. However, since no significant improvement could be found for these programs (Strömbäck et al., 2023), it leads us to ask where the mismatch occurs. Are students passing courses without grasping what educators believe they must? Is it a retention issue? Of course, it is prudent to consider that the answers provided for one course are not necessarily applicable to other courses. The Concurrent

Programming course will have higher requirements of students' grasp of, for example, indirection than the course in Algorithms and DS. But we can also see the inverse wherea course with high requirements for indirection, like Concurrent Programming, precedes a course with low requirements, like Web Programming, and the prerequisite knowledge is not considered to be good by the teacher.

5.7 Differences Between Concepts

The absence of statistically significant differences (Section 4.1.1) for the tests that we refer to as *Differences between concepts* (Section 3.5.1) does not imply that instructors' perceptions are identical or that no meaningful variation exists. Rather, it indicates that the observed differences were not large or consistent enough, given the sample size and measurement approach, to yield statistical evidence for rejecting the null hypothesis. Considering the exploratory nature of the study and the relatively small number of respondents, the Welch's t-tests were primarily included to provide an additional descriptive perspective on the survey data. However, we decided to report these non-significant results, as doing so contributes to transparency and completeness.

5.8 Limitations and Future Work

This study has several limitations that shape the interpretation of its findings.

Collected data is self-reported by the instructors. Self-reported data are valuable for capturing perceptions and experiences, but they are also subject to several constraints. Instructors may differ in how they interpret survey scales, recall past teaching practices, or assess students' understanding. Additionally, instructors' perceptions of prerequisite knowledge, assignment difficulty, and conceptual mastery is likely influenced by local and/or individual norms and expectations. As a result, the patterns observed here should be understood as representing reported perceptions, even though CS instructors at this institution probably share many expectations.

Further, the sample size is small, reflecting the limited number of instructors who teach courses for these programs that match the criteria for inclusion in this study. Small sample sizes restrict statistical power and make it difficult to detect small differences. Because the responses represent a relatively small number of individuals across three programs, the findings should not be interpreted as representative of all instructors. Rather, they provide insight into how these instructors describe their experiences. Still, the comparisons that showed statistical differences had small p-values, indicating strong evidence against the null hypothesis. Effect sizes (Cohen's d) ranged from medium to very large, suggesting substantial real-world differences between the categories measured

While the concepts themselves were defined for the instructors in this study, the Likert scales were open to interpretation. What a certain instructor interprets as explicitly teaching a concept may thus differ from another instructor's view. Nevertheless, these instruc-

tors are members of the same institutions, and it is reasonable to assume that they likely have a common interpretation of the scales. In deploying this study to a wider context, clearly defining the scales would be important.

Future research could address these limitations by incorporating multiple sources of data to triangulate instructors' perceptions with data that is not self-reported. One avenuewould be to analyze course materials, such as syllabi, lecture materials, assignments, and assessment rubrics, to identify where and how the subtle concepts appear and are addressed in the curriculum. Such analysis could help clarify whether students are exposed to the concepts, how they are reinforced, assessed, and how these align with the perceptions reported by teachers. Further depth in instructors' experiences could be unveiled through interviews to identify how expectations are communicated, conceptual difficulty is identified, and addressed.

Expanding this study's approach to additional institutions would improve external validity. Multi-institutional studies are recommended in CS eduction research to account for institutional variation (McCracken et al., 2001). A multi-institutional approach could offer a broader view of how subtle concepts are taught, reinforced, and assessed in different curricular structures. A more diverse dataset would also support stronger statistical analysis and clearer comparisons between concepts.

Overall, while the present study provides a descriptive starting point for understanding how instructors perceive subtle concept within their courses and their perceptions of development across multiple courses, future work with broader, deeper, or other methods can offer a more comprehensive picture of how these concepts are actually taught, learned, and assessed over time.

6. Conclusion

Investigating concepts that benefit from being taught in many courses during students' academic journey is challenging. The approach in this study shows promise but needs to be validated, and we suggest a deeper investigation of a subset of the courses investigated, looking closer at the course material for those courses to form a clearer connection between the data collected through the survey, the activities undertaken by students, and how those activities relate to subtle concepts.

While Figs. 4, 6 and 8 show that subtle concepts are taught early in each CS program, this teaching drops off in the latter parts of students' journey. This, while not surprising, diverges from the recommendation for teaching subtle concepts (Fisler et al., 2017).

Subtle concepts are not explicitly graded to the same extent as students are required to understand them when doing assignments, nor do courses require as firm a grasp of the concepts for students to achieve a passing grade in courses. This could mean that adequate attention is not drawn to the subtle concepts (Hewett, 2005; Fincher et al., 2020). Explicit grading of the concepts is mostly limited to programming language courses, which does indicate that there are issues with alignment (Biggs and Tang, 2015; Biggs, 1996). Thus, at

least part of the lack of improvement found for these programs previously (Strömbäck et al., 2023) can be explained by these findings.

The significant difference between students' prerequisite grasp of subtle concepts and the required grasp of the concepts to solve assignments in courses is a potential cause for concern. But this difference needs to be studied further and validated. We suggest investigating whether this difference harms the learning process for students. One wayof doing this would be to interview teachers, teaching assistants, and/or students to see whether the difference represents a challenge and opportunity for improvement or a barrier for progression.

While some findings in this paper need to be validated, we suggest the study process described in this paper as an approach to better understand the development of these types of concepts.

While teachers are encouraged to use their own creativity to address students' misconceptions (Kaczmarczyk et al., 2010), we end this paper with an actionable suggestion. Be wary when thinking that subtle concepts are implicitly graded in a course. Teachers think students need a good grasp of subtle concepts to engage with assignments and to achieve a passing grade in many courses but grade them less than explicitly. At the same time, few courses consider students to have a good grasp of the concepts in the beginning of courses. This would indicate that teachers either have very different ideas of what constitutes a good grasp of subtle concepts, or students are finding a way around learning these concepts. If we want students to acquire knowledge that is best taught across multiple courses, we need students to actively engage in organizing that knowledge (Perkins, 1999).

References

Armoni, M. (2013). On Teaching Abstraction in CS to Novices. Journal of Computers in Mathematics and Science Teaching 32, 3 (July 2013), 265–284. https://www.learntechlib.org/p/41271

Bayman, P. & Mayer, R.E. (1983). A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements. Commun. ACM 26, 9 (sep 1983), 677–679. doi:10.1145/358172.358408

Biggs, J. (1996). Enhancing teaching through constructive alignment. Higher Education 32, 3 (01 Oct 1996), 347–364. doi:10.1007/BF00138871

Biggs, J. & Tang, C. (2015). Constructive Alignment: An Outcomes-Based Approach to Teaching Anatomy. Springer International Publishing, Cham, 31–38. doi:10.1007/978-3-319-08930-0_4

Brabrand, C. & Dahl, B. (2007). Constructive Alignment and the SOLO Taxonomy: A Comparative Study of University Competences in Computer Science vs. Mathematics. In Proceedings of the Seventh Baltic Sea Conference on Computing Education Research -Volume 88 (Koli National Park, Finland) (Koli Calling '07). Australian Computer Society, Inc., AUS, 3–17.

Boulay, B.D., O'shea, T., & Monk, J. (1999). The black box inside the glass box: presenting computing concepts to novices. International Journal of Human-Computer Studies 51, 2 (1999), 265–277. doi:10.1006/ijhc.1981.0309

Fincher, S., Jeuring, J., Miller, C.S., Donaldson, P., Boulay, B.D., Hauswirth, M., Hellas, A., Her-

mans, F., Lewis, C., Mühling, A., Pearce, J.L. & Petersen, A. (2020). Notional Machines in Computing Education: The Education of Attention. In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (Trondheim, Norway) (ITiCSE-WGR '20). Association for Computing Machinery, New York, NY, USA, 21–50. doi:10.1145/3437800.3439202

Fisler, K., Krishnamurthi, S. & Wilson, P.T. (2017). Assessing and Teaching Scope, Mutation, and Aliasing in Upper-Level Undergraduates. In Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (Seattle, Washington, USA) (SIGCSE '17). Association for Computing Machinery, New York, NY, USA, 213–218. doi:10.1145/3017680.3017777

Haglund, P., Mannila, L., Strömbäck, F. & Berglund, A. (2024). Grasping the Unseen: TA Insights into Teaching Subtle Concepts in Computer Science. In Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1 (Milan, Italy) (ITiCSE 2024). Association for Computing Machinery, New York, NY, USA, 157–163. doi:10.1145/3649217.3653601

Haglund, P., Strömbäck, F. & Mannila, L. (2021). Understanding Students' Failure to use Functions as a Tool for Abstraction – An Analysis of Questionnaire Responses and Lab Assignments in a CS1 Python Course. Informatics in Education 20, 4 (2021), 583–614. doi:10.15388/infedu.2021.26

Hazzan, O. (2008). Reflections on Teaching Abstraction and Other Soft Ideas. SIGCSE Bull. 40, 2 (June 2008), 40–43. doi:10.1145/1383602.1383631

Hewett, T.T. (2005). Cognitive factors in design: overview and some implications for design. In Proceedings of the 5th Conference on Creativity & Cognition (London, United Kingdom) (C&C '05). Association for Computing Machinery, New York, NY, USA, 318–321. doi:10.1145/1056224.1056287

Hill, S. (2010). Troublesome knowledge: why don't they understand? Health information and libraries journal 27 (03 2010), 80–3. doi:10.1111/j.1471-1842.2010.00880.x

Kaczmarczyk, L.C., Petrick, E.R., East, J.P., & Herman, G.L. (2010). Identifying student misconceptions of programming. In Proceedings of the 41st ACM Technical Symposium on Computer Science Education (Milwaukee, Wisconsin, USA) (SIGCSE '10). Association for Computing Machinery, New York, NY, USA, 107–111. doi:10.1145/1734263.1734299

Koppelman, H. & Dijk, B. (2010). Teaching Abstraction in Introductory Courses. In Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education (Bilkent, Ankara, Turkey) (ITiCSE '10). Association for Computing Machinery, New York, NY, USA, 174– 178. doi:10.1145/1822090.1822140

Loughlin, C., Lygo-Baker, S., & Lindberg-Sand, Å. (2021). Reclaiming constructive alignment. European Journal of Higher Education 11, 2 (2021), 119–136. doi:10.1080/21568235.2020.1816197

Lu, K.-C. & Krishnamurthi, S. (2024). Identifying and Correcting Programming Language Behavior Misconceptions. Proc. ACM Program. Lang. 8, OOPSLA1, Article 106 (apr 2024), 28 pages. doi:10.1145/3649823

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B.-D., Laxer, C., Thomas, L., Utting, I. & 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 (Canterbury, UK) (ITiCSE-WGR '01). Association for Computing Machinery, New York, NY, USA, 125–'180. doi:10.1145/572133.572137

Mirza, D., Conrad, P.T., Lloyd, C., Matni, Z. & Gatin, A. (2019). Undergraduate Teaching Assistants in Computer Science: A Systematic Literature Review. In Proceedings of the 2019 ACM Conference on International Computing Education Research (Toronto ON, Canada) (ICER '19). Association for Computing Machinery, New York, NY, USA, 31–40. doi:10.1145/3291279.3339422

Nelson, G.L., Strömbäck, F., Korhonen, A., Begum, M., Blamey, B., Jin, K.H., Lonati, V., MacKellar, B. & Monga, M. (2020). Differentiated Assessments for Advanced Courses That Reveal Issues with Prerequisite Skills: A Design Investigation. In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education (Trondheim, Norway) (ITiCSE-WGR '20). Association for Computing Machinery, New York, NY, USA, 75–129. doi:10.1145/3437800.3439204

Norman, G. (2010). Likert scales, levels of measurement and the "laws" of statistics. Advances in Health Sciences Education 15, 5 (2010), 625–632. doi:10.1007/s10459-010-9222-y

Perkins, D. (1999). The many faces of constructivism. Educational leadership 57, 3 (1999), 6–11.

Perkins, D. & Martin, F. (1985). Fragile Knowledge and Neglected Strategies in Novice Programmers. IR85-22. Technical Report IR85-22. Educational Technology Center, Cambridge, MA, USA. https: //eric.ed.gov/?id=ED295618

Piaget, J. (2003). The psychology of intelligence. Routledge.

Qian, Y. & Lehman, J. (2017). Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. ACM Trans. Comput. Educ. 18, 1, Article 1 (oct 2017), 24 pages. doi:10.1145/3077618

Reimer, Y.J., Coe, M., Blank, L.M. & Braun, J. (2018). Effects of Professional Development on Programming Knowledge and Self-Efficacy. In 2018 IEEE Frontiers in Education Conference (FIE). 1–8. doi:10.1109/FIE.2018.8659041

Sorva, J. (2013). Notional machines and introductory programming education. ACM Trans. Comput. Educ. 13, 2, Article 8 (July 2013), 31 pages. doi:10.1145/2483710.2483713

Strömbäck, F., Haglund, P., Berglund, A., & Berglund, E. (2023). The Progression of Students' Ability to Work With Scope, Parameter Passing and Aliasing. In Proceedings of the 25th Australasian Computing Education Conference (Melbourne, VIC, Australia) (ACE '23). Association for Computing Machinery, New York, NY, USA, 39–48. doi:10.1145/3576123.3576128

Strömbäck, F., Mannila, L., Asplund, M. & Kamkar, M. 2019. A Student's View of Concurrency -A Study of Common Mistakes in Introductory Courses on Concurrency. In Proceedings of the 2019 ACM Conference on International Computing Education Research (Toronto ON, Canada) (ICER '19). Association for Computing Machinery, New York, NY, USA, 229–'237. doi:10.1145/3291279.3339415

Sullivan, G.M. & Artino Jr., A.R. (2013). Analyzing and Interpreting Data From Likert-Type Scales. Journal of Graduate Medical Education 5, 4 (Dec. 2013), 541–542. doi:10.4300/JGME-5-4-18

Acknowledgments

We would like to thank the teachers in the Department of Computer Science and Department of Electrical Engineering at Linköping University for taking the time to participate in this study.

A. Syllabi

In this section the syllabi for each course is presented.

A.1. CS Bachelors

- Intro Ada/C++ -https://studieinfo.liu.se/kurs/TDIU08
- Intro Assembler -https://studieinfo.liu.se/kurs/TSIU05
- OOP C++ -https://studieinfo.liu.se/kurs/TDIU20
- C++ Project -https://studieinfo.liu.se/kurs/TDDI82
- Algorithms and DS (C++) -https://studieinfo.liu.se/kurs/TDDI16
- Concurrent Programming (C) -https://studieinfo.liu.se/kurs/TDIU16
- Project -https://studieinfo.liu.se/kurs/TDDI17
- Web Programming -https://studieinfo.liu.se/kurs/TDDD97

A.2. CS Masters Software

- Python Part 1 -https://studieinfo.liu.se/kurs/TDDE23
- Python Part 2 -https://studieinfo.liu.se/kurs/TDDE24
- Assembler Intro -https://studieinfo.liu.se/kurs/TSEA28
- App Project (Python) -https://studieinfo.liu.se/kurs/TDDD80
- Java -https://studieinfo.liu.se/kurs/TDDD78
- Algorithms and DS (C++) -https://studieinfo.liu.se/kurs/TDDD86
- Concurrent Programming (C) -https://studieinfo.liu.se/kurs/TDDE68
- AI (Python/Java) -https://studieinfo.liu.se/kurs/TDDD92
- Bachelor Project -https://studieinfo.liu.se/kurs/TDDD96

A.3. CS Masters Hardware

- Assembler Intro -https://studieinfo.liu.se/kurs/TSEA22
- Java -https://studieinfo.liu.se/kurs/TDDE30
- Assembler 2 -https://studieinfo.liu.se/kurs/TSEA83
- Assembler 3 -https://studieinfo.liu.se/kurs/TSEA29
- Bachelor Project -https://studieinfo.liu.se/kurs/TDDD96

B. The Survey

Course design and subtle concepts in Computer Science

Thank you for participating in this study of how subtle concepts are taught throughout a number of computer science programs at this university. Subtle concepts, in the context of this survey are: indirection, references, scope, and parameter transfer. Your participation is voluntary, and you have the right to withdraw at any point. While we will ensure confidentiality by not identifying participants or disclosing specific course codes in our publication, it will be necessary to detail the courses involved for the sake of clarity for the readers. Consequently, complete anonymity cannot be guaranteed.

* *Required*

1. Write down the course code and name of your course: *

2. For students in this course, is programming a central part of achieving the course objectives? *
   - ○ Yes
   - ○ No
   - ○ Do not know

3. Please leave a short description of the programming related activities in the course: *

4. This question relates to students' understanding of four programming concepts when they start taking your course.
Indirection: In this survey you should consider indirection in the form of manipulating or accessing values through the use of pointers as well as dynamic dispatch of function calls.
References: In this survey you should consider references to be the use of aliases for manipulating or accessing values.
Scope: The scope of name binding, that is the part of the source code in which the name binding of an entity (such as a variable) is valid.
Parameter Passing: The transfer of parameters to functions, and its semantics, with any qualifiers that may be applicable.
When first starting to take this course students have a good understanding of: *

	Strongly agree	Agree	Neither agree nor disagree	Disagree	Strongly disagree
indirection	◯	◯	◯	◯	◯
references	◯	◯	◯	◯	◯
scope	◯	◯	◯	◯	◯
parameter passing	◯	◯	◯	◯	◯

5. Please briefly describe your view of students' prior knowledge relating to these concepts:

6. This question relates to how the four programming concepts are integrated in your course and how explicitly they are taught.
The following concept is taught in the course: *

	Explicitly	Somewhat Explicitly	Implicitly	Somewhat Implicitly	Not at all
indirection	◯	◯	◯	◯	◯
references	◯	◯	◯	◯	◯
scope	◯	◯	◯	◯	◯
parameter passing	◯	◯	◯	◯	◯

7. Please briefly describe the extent to which the concepts are taught as well why they are taught to that extent.

8. This question relates to what extent students need to understand the four concepts when solving assignments (such as labs, seminars, and exams) in the course. When working with assignments students need to have a good understanding of this concept in order to solve the assignment: *

	Strongly Agree	Agree	Neutral	Disagree	Strongly disagree
indirection	◯	◯	◯	◯	◯
references	◯	◯	◯	◯	◯
scope	◯	◯	◯	◯	◯
parameter passing	◯	◯	◯	◯	◯

9. Please briefly describe students need to understand these concepts when working with assignments in the course

10. This question relates to how explicitly the four concepts are graded by course staff.
The following concept is graded by course staff: *

	Explicitly	Somewhat Explicitly	Implicitly	Somewhat Implicitly	Not at all
indirection	◯	◯	◯	◯	◯
references	◯	◯	◯	◯	◯
scope	◯	◯	◯	◯	◯
parameter passing	◯	◯	◯	◯	◯

11. Please briefly describe how explicitly these concepts are graded in the course

12. Please briefly describe how explicitly these concepts are graded in the course
Considering how the course is designed and structured, how good does a students understanding of these four programming concept have to be in order to achieve a passing grade in the course? *

	Very Good	Good	Fair	Poor	Very poor
indirection	◯	◯	◯	◯	◯
references	◯	◯	◯	◯	◯
scope	◯	◯	◯	◯	◯
parameter passing	◯	◯	◯	◯	◯

13. Please provide a brief description of how well students need to understand the concepts in order to achieve a passing grade in the course.

14. If you have any final thoughts you would like to share you may do so here: