Master thesis
Information Sciences

RADBOUD UNIVERSITEIT

The Coderclass Decoded

Analyzing a new extracurricular course

Author:
Jory van Keulen
4038053

Main supervisor/assessor:
Erik Barendsen
e.barendsen@cs.ru.nl

Second assessor:
Sjaak Smetsers
s.smetsers@cs.ru.nl

Abstract

This study investigates a new secondary school computer science elective course in Amsterdam, the Netherlands, called the Coderclass. Because this elective course has a curriculum that deviates from the standard Dutch computer science curriculum, a case study is performed into the curriculum's content, learning objectives and student attitudes of the first year of the elective course.

A conceptual analysis of the Coderclass first year modules is performed and compared to other computer science curricula. In this analysis, it becomes apparent that several categories are much more focussed on in the Coderclass in comparison to peer curricula, such as programming and modeling, while others are underrepresented, such as intelligence and society.

Learning objectives are then extracted from module goals and assignments and are found to focus on cooperation, debugging and testing, decomposing, designing, generalization, initiative and planning, modeling, understanding and writing a program, and understanding general computer science concepts. The learning goals are then compared to the core 2016 Dutch computer science curriculum to point out that the Coderclass focuses much more on the core domain of programming while focusing less on the core domains of interaction and architecture. It also becomes clear that the learning objectives found are in line with a widely accepted definition of computational thinking, and that Coderclass students will learn to think like a computer scientist.

Furthermore, student attitudes on the Coderclass are analyzed through learner reports in which the general positivity on student confidence, satisfaction and judgement on the elective course is expressed by Coderclass students.

Finally, a new module Entrepreneurship-0 is developed as a contribution to the Coderclass curriculum, and specifics of this module are discussed.

Contents

Preface

As the world moves further into the digital age, the education of its inhabitants must also be updated to match this progression. As my own secondary school computer science course was somewhat outdated, even for its time, it was my great interest to see new initiatives on various computer science courses, and I have chosen to commit my master's thesis on a brand new elective course in Amsterdam by performing an in-depth analysis on "the Coderclass".

The research for this thesis was performed during an analysis of the 2016-2017 Coderclass academic year of the Metis Montessori Lyceum (MML) in Amsterdam. The thesis itself was written and improved throughout 2017-2019.

During the research, I have collaborated with the MML and have made several visits to their Coderclass in Amsterdam. To perform the analysis itself, I have been given access to many documents and sources that are used in teaching the students of the Coderclass, such as the course wiki containing the course modules.

Apart from the analysis performed for this research, I have also been asked to contribute to the Coderclass course by creating a new module. This module is the starting module for the course involving entrepreneurship and is also mentioned further on in the thesis.

I want to thank Erik Barendsen, who has been a great supervisor and has helped me in performing my research and writing this thesis. Furthermore, I would also like to thank Hakan Akkas, one of the founders of the Coderclass, who has shown me great hospitality and has helped me attain the data needed to perform this study. Finally, I would like to thank the MML and the Coderclass students who have allowed me to observe several Coderclass classes and have particularly helped in performing an attitude outcome analysis.

Chapter 1

Introduction

Like anywhere else in the world, Computer Science (CS) is an essential topic in the Netherlands. However, the Netherlands only features a formal CS course in upper secondary school, which does not include lower secondary school, and is in contrast to other countries. To close the gap between the Netherlands and other countries on this subject, more and more CS educational initiatives are arising, even in the junior classes of secondary education. The main problem of this is that secondary schools create these new CS courses on their initiative, which can result in a lot of entirely different curricula that are often quite different from the formal Dutch CS curriculum. Furthermore, schools often do not know for sure if they are doing it right. All of this leads up to the necessity of studying these new initiatives, and the Coderclass is no exception. But how does one study and compare wholly new and different CS curricula? How do we 'decode' the Coderclass?

This thesis aims to perform a case study on such a new CS initiative. By performing a case study into a new CS elective course called "the Coderclass", using different tools such as conceptual analysis and attitude learner reports, one possible method of studying and comparing CS curricula is described. The Coderclass features a new CS curriculum by its own design and is performed at the Metis Montessori Lyceum (MML) in Amsterdam. Coderclass students start their CS course as early as their first year in secondary school. This study will also help the Coderclass in figuring out where they stand regarding their new curriculum by analyzing and discussing their specific course materials and student attitude outcomes. The result of this will help the newly founded Coderclass determine how it holds up compared to the general Dutch CS course and other CS curricula, and possibly provide a method of analyzing new CS curricula by providing results of different research methods in which some may be more useful than others. Finally, this study will also feature a contribution to the Coderclass in the form of

a new module as part of the research.

1.1 Structure of this thesis

This thesis will begin with various background material on educational CS research that is relevant to this study (chapter 2). In chapter 3, the aim of the study is explained, and research questions are given, after which the methodology of performing the study is given in chapter 4. In chapter 5, the results of the study are shown extensively. Chapter 6 answers the research questions in the form of conclusions. Chapter 7 features the specifics of the contribution that was made to the Coderclass as part of the research. Finally, a reflection on the thesis is made, and a discussion on the methods is featured in chapter 8.

Chapter 2

Background

This chapter features background information on various topics that are of importance for this research. To discover specifics of the Coderclass curriculum (such as content and learning objectives), their teaching and learning environment is first mapped out and then used as an oversight to explain background information on other important topics related to this study.

2.1 The curricular spiderweb

To determine the specifics of a curriculum (such as learning objectives), it helps first to map out the different areas of the curriculum. An often-used method of doing so is the "curricular spiderweb" of Van den Akker (2004). This spiderweb allows one to analyze a curriculum from the top down by splitting it into several different aspects. It begins with the rationale of the curriculum, which forms the center of the spiderweb, to which nine strands of educational subdomains are connected. Ideally, each one of these subdomains is also connected to the others, thus forming a spiderweb (see figure 1). These subdomains each aim to answer a specific question about the curriculum, as shown in table 1. Several of these strands (in particular Content and Aims & Objectives) are of importance to this study as they will help determine the Coderclass curriculum specifics.

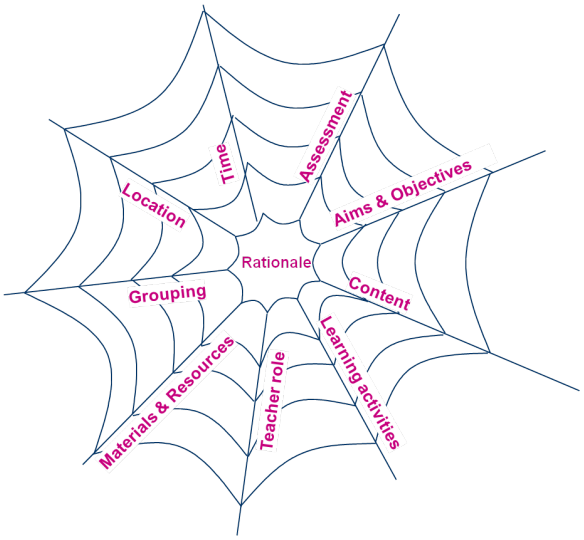

Figure 1: Van den Akker's curricular spiderweb

Rationale	Why are they learning?
Aims & Objectives	To what aim are they learning?
Content	What are they learning?
Learning activities	How are they learning?
Teacher role	What is their teacher's role with learning?
Materials & Resources	With what are they learning?
Grouping	With whom are they learning?
Location	Where are they learning?
Time	When are they learning?
Assessment	How is their learning assessed?

Table 1: Van den Akker's curricular spiderweb: sub domains and the questions they can answer.

2.2 Content

To analyze a curriculum according to the curricular spiderweb, the spiderweb subdomain of *content* is of the highest importance. This domain explains the topics and concepts that are taught in an educational course. Thus, a conceptual analysis of the curriculum's content is required.

Such an analysis can be performed in many different ways. The methodology used in this study is an adaptation of Barendsen and Steenvoorden (2016). In their research, several curricula were investigated through conceptual analysis. These are the former Dutch CS curriculum, the 2012 French CS curriculum, the 2012 Computing at School (CAS) guidelines, the 2011 CSTA standards and the new 2016 Dutch CS curriculum.

During these analyses, CS subjects were classified based on knowledge areas of the CS curriculum. These knowledge areas are then clustered in smaller categories. The course and its documents are scanned for codes that relate to these categories, after which they are used to get a global overview of occurrences of codes in each category. The outcomes can then be used to see which concepts occur most and which concepts are not covered by the course, and can also be used to compare different courses. The results of these analyses show that data, architecture, networking, algorithms, and engineering cover the most significant parts of the studied specifications, but that there are differences in emphasis. An overview of the concepts and knowledge categories is given in table 2, and the outcome of the research is given in figure 2.

Knowledge category	Included ACM/IEEE knowledge areas
Algorithms	Algorithms and complexity (AL)
	Parallel and distributed computing (PD)
	Algorithms and design (SDF/AL)
	Remark: concepts about data structures are covered by *Data*
Architecture	Architecture and organization (AR)
	Operating systems (OS)
	System fundamentals (SF)
Modeling	Computational science (CN)
	Graphics and visualization (GV)
Data	Information management (IM)
	Fundamental data structures (SDF/IM)
Engineering	Software engineering (SE)
	Development methods (SDF/SE)
	Remarks: also contains ideas on collaboration; concepts without an engineering component are covered by programming
Intelligence	Intelligent systems (IS)
Mathematics	Discrete structures (DS)
Networking	Networking and communication (NC)
Programming	Programming languages (PL)
	Platform based development (PBD)
	Fundamental programming concepts (SDF/PL)
Security	Information assurance and security (IAS)
	Remark: concepts about privacy are covered by society
Society	Social issues and professional practice (SP)
Usability	Human-computer interaction (HCI)

Table 2: Knowledge categories.

CSTA

1. Algorithms (44)
2. Engineering (40)
3. Architecture (37)
4. Society (30)
5. Networking (27)
6. Programming (25)
7. Data (23)
8. Security (13)
9. Modeling (12)
10. Intelligence (11)
11. Mathematics (8)
12. Usability (2)
13. Rest (0)

(Total: 272)

CAS

1. Algorithms (44)
2. Networking (40)
3. Architecture (38)
4. Data (33)
5. Programming (19)
6. Engineering (17)
7. Mathematics (5)
8. Security (4)
9. Society (2)
10. Intelligence (1)
11. Modeling (0)
    Rest (0)
    Usability (0)

(Total: 203)

France

1. Data (28)
2. Programming (15)
3. Architecture (14)
   Networking (14)
4. Algorithms (13)
5. Mathematics (8)
6. Society (5)
7. Engineering (4)
   Modeling (4)
8. Intelligence (2)
9. Rest (1)
10. Security (0)
    Usability (0)

(Total: 108)

Netherlands 2007

1. Architecture (13)
2. Data (12)
3. Engineering (10)
4. Networking (4)
   Rest (4)
5. Programming (3)
6. Usability (3)
7. Modeling (2)
8. Security (1)
9. Algorithms (0)
   Intelligence (0)
   Mathematics (0)
   Society (0)

(Total: 52)

Netherlands 2016 (core)

1. Programming (18)
2. Engineering (17)
3. Data (11)
4. Society (10)
5. Architecture (9)
6. Security (7)
7. Algorithms (6)
8. Usability (3)
9. Networking (2)
10. Intelligence (0)
    Mathematics (0)
    Modeling (0)
    Rest (0)

(Total: 83)

Netherlands 2016 (complete)

1. Programming (22)
2. Architecture (19)
   Society (19)
3. Data (18)
   Engineering (18)
   Usability (18)
4. Security (16)
5. Algorithms (14)
6. Networking (11)
7. Modeling (7)
8. Mathematics (4)
9. Intelligence (3)
10. Rest (0)

(Total: 169)

Figure 2: Lists of knowledge categories for each curriculum document, sorted from most to least occurring concepts. The number of concept occurrences in each category is displayed between parentheses. The total number of concept occurrences in the document is given at the end of each list.

2.3 Aims & Objectives

The second strand of the curricular spiderweb that is of importance to this study is the *Aims & Objectives* strand. This particular strand of the spiderweb focuses on answering the question "To what aim are they learning?". While *Content* specifies the what, *Aims & Objectives* focusses more on the why. This strand contains curriculum specifics such as learning objectives but can also include aims that are more focused on the student's attitude of the subjects. One particular learning objective that is of importance to new CS curricula is computational thinking. Computational thinking and student attitudes are first addressed before heading further into the study.

2.3.1 Computational thinking

A particular learning objective that is often mentioned in a present-day CS curriculum is computational thinking (CT). CT is a term that took off in the CS community when an article about computational thinking was published in an ACM Communications journal by Jeanette Wing. According to Wing (2006), CT is a way for humans to solve problems by reformulating a difficult issue into an easier one that we can solve by means of thinking recursively and using recognition, interpreting, abstraction conceptualization, and analysis to solve a problem, and not just problems in CS; CT can be used to solve problems in any context. The general definition of CT given by Wing and adhered to by many is *thinking like a computer scientist*, and that "computational thinking involves solving problems, designing systems, and understanding human behavior, by drawing on the concepts fundamental to computer science" (p. 33). According to Wing, CT should be taught to everyone, not just computer scientists, and integrated into other disciplines besides CS. Since this article in 2006, the term has increasingly been used in the shaping of courses for students in primary and secondary education.

CT often appears in CS curricula, intended or not, by making students think like a computer scientist. This can be achieved through various methods, one of which is programming behind a computer. Other methods involved unplugged ways of teaching, such as making a student solve an algorithmic problem on paper. The usability and necessity of teaching CT in secondary education is still a source of many discussions today.

Researchers and CS educators now work with several descriptions of CT, with the value of *abstraction* being CT's keystone (Grover & Pea, 2013) or in other words, a "focused approach to problem-solving using thought processes that utilize abstraction, decomposition, algorithmic design, evaluation, and generalizations" (Selby & Woollard, 2013). According to Grover and Pea (2013), the following elements are now widely accepted as comprising CT,

and form the basis of curricula that aim to support its learning and assess its development.

- Abstractions and pattern generalizations (including models and simulations)
- Systematic processing of information
- Symbol systems and representations
- Algorithmic notions of flow of control
- Structured problem decomposition (modularizing)
- Iterative, recursive, and parallel thinking
- Conditional logic
- Efficiency and performance constraints
- Debugging and systematic error detection

Grover and Pea's list of CT elements will also be used as the definition of CT in this study.

2.3.2 student attitudes

An important aim of an educational course in recent days is to change the way a student thinks about the subjects covered in the course and for the students to be able to use what they learn in their everyday lives. Many sources have already stressed the importance of student's attitudes towards modern science curricula and learning outcomes, such as Council et al. (1996) and Binkley et al. (2012).

One method of modeling these student attitudes that is used in this study are the learner reports created by De Groot (1980). According to De Groot, learning outcomes can be characterized as knowledge and skills that have been attained by students in a course. These are things students can speak or write about, and thus learning outcomes can be translated into a form in which a student can report their learning experiences. De Groot classifies two different kinds of learning: learning about rules and exceptions, and learning about the world around you and yourself, thus objectively learning about rules and exceptions about the world, and rules and exceptions about yourself. A model of this characterization is given in figure 2 (Eijk, 1984). In this model, students are asked about their learning experiences. The learning reports will help determine the connections, student attitudes, and concepts discovered and used by the students in any given curriculum.

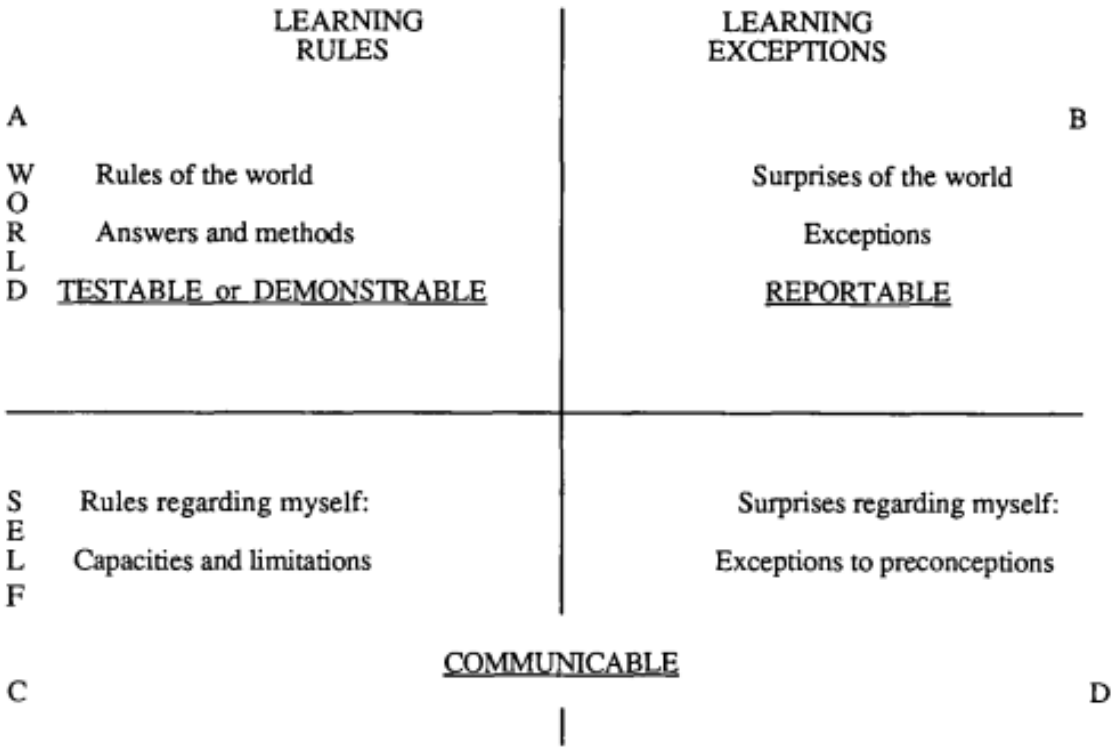

Figure 3: Classification of all Learning Objectives According to De Groot (From: Van Eijk, 1984).

As mentioned before, learner reports point out student attitudes as a way to judge learning outcomes, which is not a new concept. Klop (2008) divides the attitudes into 3 separate components:

- Cognitive component: evaluation using opinions and knowledge
- Affective component: emotions and how people feel about things
- Behavioral Component: behavioral intentions

The motivation of students is assumed to have an impact on these attitudes and whether they are positive or negative. The motivational ARCS model of Keller (1984) categorizes motivation into 4 factors: Attention, Relevance, Confidence and Satisfaction (ARCS). These motivational factors can then be connected to the attitude components by Klop (2008). The Satisfaction factor appeals to the affective attitude component, whereas Relevance and Confidence contribute to the cognitive component. Klop's components and

Keller's ARCS model will form the basis of learner report analysis for this study.

Chapter 3

Aim of the study

The primary aim of this research is to study the *Content* and *Aims & Objectives* strands of the curricular spiderweb for the first year of the Coderclass. While doing this, the aim is to determine the course content, learning objectives, and student attitude outcomes. This is done by performing a conceptual analysis in order to find out the content and learning outcomes of the curriculum, and by performing an analysis on learner reports filled in by the Coderclass students to discover student attitude outcomes.

Secondary aims of this research include comparing the results of the conceptual analysis to different CS curricula (including the general 2016 Dutch CS curriculum) as previously analyzed by Barendsen and Steenvoorden (2016), and finding out in what way (if any) CT is included in the learning objectives of the Coderclass.

More context on the Coderclass itself and the current situation of Dutch CS curriculum will first be given to achieve these goals.

3.1 Context of the study

3.1.1 The Metis Montessori Lyceum & The Coderclass

The MML in Amsterdam is the home of the new project "Coderclass" (https://coderclass.nl/), an elective course in which motivated secondary education students are introduced to CS through a weekly 5-hour program. During these 5 hours, they can work at their own pace to learn about several areas of CS. Once they have enough knowledge about a particular subject, they are awarded a badge. This badge then allows them to work on a practical issue (often related to actual businesses) with a group of fellow students who have also attained this badge. During this practical assignment, they use what they have learned to create something in a group environment. This is done using a "light version" of the SCRUM developing

method. The Coderclass also has strong ties to several IT businesses, who provide them with realistic challenges and guest lectures for the students. The Coderclass is a rather new course and still relies much on trial-and-error to improve the quality of the course.

When the analysis for this study was performed, the Coderclass had just ended their first year of the course. After the first year of core subject teaching and having attained the minimum yearly program each year, students gain more options to develop skills and specialize in areas they are most passionate about as they can choose the modules they take in the course. Even in their first year, students gain some opportunity to choose an additional, optional module they are interested in to add to their education. However, as these optional modules and specializations are not part of the intended basic direction of the curriculum, they are not included in the analysis.

Figure 4: The Metis Montessori Lyceum in Amsterdam

The Coderclass through the curricular spiderweb
Several visits to the MML were made during the study, in which observations of classes were performed. During these visits, additional information about other strands of the curricular spiderweb was gained by performing semi-structured interviews with the staff of the Coderclass. By combining these

additional sources of information, the Coderclass can be viewed from the top-down through the curricular spiderweb (with the exception of *Aims & Objectives* and *Content*, as these are the main focus of the study and are analyzed further on):

- **Rationale: Why are they learning?**
  Coderclass students are learning about CS at an early age, so they get used to CS practical matters that they could benefit from in their life, and possibly in their professional careers.

- **Learning activities: How are they learning?**
  Students in the Coderclass learn about CS through reading and making assignments about a specific topic in modules, after being given an introductory lecture in class by teachers or guest lecturers. After they have mastered this material, they are given a badge to show their mastery. Then, they gain practical experience by using the material they have learned in one or more practical assignments that they perform in bigger groups. These practical assignments often required skills learned in multiple modules.

- **Teacher role: What is their teacher's role with learning?**
  The teachers have a huge part in creating the course from scratch by creating and editing the modules if needed. They are also tasked with explaining the theory and assignments amongst the students and helping them when they need guidance. They are responsible for the assessment of the modules and (practical) assignments done by the students.

- **Materials & Resources: With what are they learning?**
  The Coderclass uses a wiki to spread the theoretical and practical parts of the course, which students open on laptops they bring themselves, or on computers that are present in the computer lab. The students also have access to a forum environment (Out-Of-Bounds) in which they can ask questions regarding the course. Fellow students and teachers can then help them with their questions.

- **Grouping: With whom are they learning?**
  Coderclass students start each module by doing assignments on their own to get acquainted with the material. When the teachers deem they have sufficient knowledge of the material, a badge is awarded to them for possessing the required knowledge. Then, they are divided into groups to use the knowledge they gained in a larger practical assignment.

- **Location: Where are they learning?**
  The Coderclass course is given in a special computer lab inside the

MML. Some of the classes are also given on location at specific CS-related businesses (mostly by guest lecturers), where the Coderclass students perform practical assignments.

- **Time: When are they learning?**
  The Coderclass students are required to be present on the MML for 5 hours each week. The schedule differs according to the school's lecture rosters. Homework is also given to the students, in particular for the theoretical parts of the course. The homework is mostly done from home, but can also be done during class if the student has enough time. Practical assignments are mostly performed during classes (which means students can work together and ask for guidance if necessary), but students can also work on this from their homes.

- **Assessment: How is their learning assessed?**
  Their teachers assess the students' work and assignments. Whenever a student is finished with a specific module, they can request a badge for it. The teachers will then assess the work done by the student in order to see if they have sufficiently finished all assignments and gained the skills required to achieve the badge. When they deem the student has done so, the badge for that specific module is awarded. Practical assignments are assessed similarly. When a group of students is finished with their work, the completed work can be assessed by their teachers, who will grade them for their work.
  The Coderclass has no final exam or tests, assessment of students is primarily done by the badge system explained above, and the assessment on the practical assignments made.

3.1.2 Dutch CS Education

In the Netherlands, CS is an elective subject. There has long been a CS curriculum for students in upper secondary education. This curriculum was established in 1998, had some minor adjustments in 2007 (Schmidt, 2008) and had a complete reform in 2016. The Dutch government decides what curriculum is used for every subject during primary and secondary school (including CS). The government often relies in part on curriculum research performed by scientists to make these decisions.

After elementary school (which, in the Netherlands, is completed at 12 years old), students move on to one of three types of secondary education. Only two of these (havo and vwo) offer CS as a subject and will be discussed, as the third does not typically involve a longer CS curriculum. The first of the two (havo; in Dutch: hoger algemeen vormend onderwijs) is a curriculum of five years (grades 7 through 11) that focuses on preparing students for professional education. The second of the two (vwo; in Dutch: voorbereidend

wetenschappelijk onderwijs) is a curriculum of 6 years (grades 7 through 12) and prepares students for further education at a university. The Dutch government decides what curriculum is used for every subject and how they are assessed. A curriculum is formulated as a set of learning objectives that students should have mastered on the completion of the subject. These curricula are often quite abstract descriptions of the intended learning outcomes, as only the 'what' is decided by the government and the 'how' is left to the schools (Barendsen, Grgurina, & Tolboom, 2016). To help build a curriculum from the core learning objectives, it is also necessary to build and improve smaller domains of the curriculum. Because of this, learning objectives are often grouped into domains. Each of these domains is subdivided into one or more subdomains and is then specified by a specific learning objective.

The 2007 version of the CS curriculum held 18 learning objectives (Tolboom, Grgurina, et al., 2008) and was presented as shown in table 3.

Domain	Subdomains
A: Informatics in perspective	Science and technology, society, study and career, the individual
B: Terminology and skills	Data representation in a computer, hardware, software, organizations
C: Systems and their structures	Communication and networks, operating systems, systems in practice, development of information systems, information flow, information analysis, relational databases, human-computer interaction, system development life cycle
D: Usage in a context	

Table 3: Domains and subdomains in CS, 1998 curriculum after 2007 revision (Barendsen et al., 2016)

Despite several evaluations of the curriculum over the years, there have never been major changes to it. This neglect led to a new study as ordered by the Ministry of Education into the teaching of CS and its curriculum (Tolboom, Krüger, & Grgurina, 2014). By the end of this new study, it was confirmed that the situation was worrying and in need of an update. This led to the decision made by the Ministry of Education to reform the curriculum. The new, reformed curriculum was adopted as the new informatics curriculum in 2016.

The new 2016 curriculum's core consists of a set of skills that address CS

specific skills, general scientific skills, and technical skills. Besides the skills, the core curriculum also consists of five knowledge domains and finally, a set of elective themes. An overview of these domains and themes is given in table 4.

Core Curriculum	Elective themes
Domain A Skills	Domain G Algorithms, computability, and logic
Domain B Foundations	Domain H Databases
Domain C Information	Domain I Cognitive computing
Domain D Programming	Domain J Programming paradigms
Domain E Architecture	Domain K Computer architecture
Domain F Interaction	Domain L Networks
	Domain M Physical computing
	Domain N Security
	Domain O Usability
	Domain P User experience
	Domain Q Social and individual impact of CS
	Domain R Computational science

Table 4: Domains of the 2016 CS curriculum (Barendsen et al., 2016)

Many Dutch universities and businesses also offer a wide range of curricula involving CS, which are all different from one another (as they are usually not influenced by the government), but are often inspired by research done into CS curricula and education.

With this in mind, this study aims to contribute to these CS curricula studies and to place the Coderclass analysis in context by comparing it to the 2016 CS curriculum (among others).

3.2 Research questions

This study will focus on discovering the conceptual content, learning objectives, and student attitude outcomes in the first year of the Coderclass. Special attention is given to computational thinking and how it is involved in the learning objectives. Furthermore, only the first year of the Coderclass is analyzed, as it the Coderclass had just finished their first year of the course by the time of the analysis done for this study. The entire six-year curriculum and the long term benefits will not be part of this analysis.

To sum up the previous information into a set of research questions, this study aims to answer the following research questions:

Conceptual content

1. What is the conceptual content of the first year of the Coderclass?
2. How does the content of the Coderclass compare to other CS curricula?

Learning objectives

3. What are the learning objectives in the first year of the Coderclass?
4. How do the learning objectives of the Coderclass compare to the Dutch Computer Science learning objectives?
5. In what way do the learning objectives of the Coderclass involve Computational Thinking?

Attitude outcomes

6. Which learning outcomes can be determined concerning Coderclass student attitudes?

Chapter 4

Methodology

An extensive analysis of the first year of the Coderclass was performed to answer the research questions. The methodology of research for each topic of this study is included in this chapter.

4.1 Conceptual content

To determine the conceptual content set out for the first year of the Coderclass, the strand of *Content* of Van den Akker's curricular spiderweb is relevant. This strand answers the question "What are they learning?".

4.1.1 Conceptual analysis

In order to find out the answer to this question, an in-depth analysis of the learning material and documents of the Coderclass was performed. These documents feature the modules that all students work within their first year, and contain the concepts they learn about and the assignments they must finish. The main goal of this analysis was to discover which concepts are taught in the Coderclass so that an accurate representation of the conceptual content in the first year can be given. This analysis began by using an adaptation of the conceptual characterization of a curriculum that was developed by Barendsen and Steenvoorden (2016). Their characterization is a conceptual analysis method used to analyze the conceptual content of a curriculum. The starting point of the method was a classification of CS subjects in terms of knowledge categories, based on the "knowledge areas" of the computing curricula. These knowledge areas were then clustered in small categories. Then, the curriculum and documents were scanned for codes that relate to these categories and then used to get a global overview of occurrences of codes in each category. The outcomes could then be used to see which concepts occur most, which concepts are not covered by the curriculum, and could also be used to compare different curricula. The

coding of these categories was done through qualitative analysis software called Atlas.ti, which helps systematically uncover and analyze issues that are hidden in data. Their method of analysis of a curriculum through coding in Atlas.ti is copied in this thesis and was used in the analysis of the Coderclass.

To code in Atlas.ti, auto-coding using specific terms related to knowledge categories as determined by Steenvoorden (2015) (see Appendix 8.1) was first used to start the analysis in a deductive manner. Then, the analysis was broadened by inductively scanning the documents for additional terms and concepts. When one of the terms was found somewhere in the documents, it was decided whether the hit would indeed relate to the specific knowledge category. This decision was made by scanning the context and omitting occurrences that cover the same concept and context as an earlier coded occurrence in order to avoid a distorted final tally. Concepts can occur multiple times in a knowledge category but are only coded if they appear with different context or meaning. This was also done by coding fragments of text that include context for these concepts, rather than simply coding every concept occurrence. Thus, the unit of analysis in this study are pieces of text covering a specific concept that relates to one of the twelve knowledge categories, rather than singling out concept occurrences while omitting most of the context. The reason for this is because this study analyses an entire educational year and covers all of the subject material that is taught in the first year, while this methodology is originally designed to analyze educational guideline documents.

The results for this analysis are concepts found in each knowledge category. They are and shown in a general oversight, after which specific concepts for each knowledge category are listed. Then, connections and specifics of each knowledge category are discussed, including several examples from the course modules.

4.1.2 Comparing the Coderclass to other CS curricula

To broaden the perspective on the discovered conceptual content in the conceptual analysis, the results of the conceptual analysis were then compared to several other CS curricula by using the data acquired by Barendsen and Steenvoorden (2016). This was done by comparing the occurrences of concepts in one curriculum to those of another, and by including visualizations to simplify the comparison. Since concepts in actual study material were compared to concepts from educational guideline documents, the focus was put on the ratio of each knowledge category in regard to the total amount of concept occurrences to see what each CS curriculum focusses on the most.

The Coderclass curriculum was compared with the Dutch 2016 CS curriculum (both the core and total curricula), the US teacher organization standards for K-12 computer science CSTA, the English Computing at School curriculum (CAS) and the French CS curriculum.

Results in this sections are visualized, after which some of the more interesting differences between the Coderclass and the other CS curricula are discussed.

4.2 Learning objectives

In order to discover the learning objectives set out for the first year of the Coderclass, the subdomain of *Aims & Objectives* of van den Akker's curricular spiderweb was used. This subdomain answers the question "To what aim are they learning?".

4.2.1 Learning objective analysis

Two different analyses have been made to discover the learning objectives in the first year of the Coderclass. First, the learning objectives of each separate module are analyzed and classified into core CS domains according to the 2016 Dutch CS curriculum domains of Barendsen et al. (2016). These learning objectives are discovered through the analysis of the following two sources:

- **The requirements for attaining the badge in each module**: Before Coderclass students start a new module, they are shown an overview of the module they are about to tackle. Among other data, this overview contains a list of requirements of what must have been done in order to adequately finish the module and achieving the badge for that module. These requirements have been analyzed and reverse engineered into a set of overall skills.
- **The assignments in each module**: To gain a better insight into the learning objectives in each module and their requirements, the assignments within the modules have been analyzed and reverse-engineered into overall skills to complement the initial learning objectives given at the beginning of the module.

To protect the content of the Coderclass modules, which is intellectual property, these lists of requirements and explicit assignments will not be explicitly shown in this thesis.

Through analysis of these sources, learning objectives have been extracted and classified in the core domains of Skills, Foundations, Information, Programming, Architecture, and Interaction (Table 4). They are only classified into core domains and not elective themes, as this analysis only focuses on mandatory modules of the Coderclass and not the elective ones. The core domains of the 2016 Dutch CS curriculum are also mandatory throughout the curriculum and give a better representation of the learning objectives than the elective themes.

Next, the same sources and the results of the previous analysis have again been analyzed to discover recurring learning objectives in order to present

the main learning objectives for the entire first year of the Coderclass (whereas the first analysis presented learning objectives per module).

4.2.2 Comparing the Coderclass learning objectives to the 2016 Dutch CS learning objectives

After extracting learning objectives from each Coderclass module, they were compared to the 2016 Dutch CS learning objectives as described by the Dutch SLO national expert center for curriculum development (Barendsen & Tolboom, 2016). Through this comparison, it is possible to see in what way the first year of the Coderclass adheres to the overall 2016 Dutch CS core curriculum and on what areas the Coderclass focuses on different learning objectives.

This was done by specifically looking into the subdomains of each of the core CS domains and reviewing how often each subdomain is present in the modules of the first year of the Coderclass, through linking the learning objectives to the subdomains (using Atlas.ti) and finally using the tally of each occurrence of the CS subdomain to see how much the Coderclass focuses on the separate subdomains of learning objectives. The tally of each subdomain is displayed, and the specifics of the comparison are discussed in the results.

4.2.3 Computational Thinking in the Coderclass

The set of recurring learning objectives that have been discovered through the second, overall Coderclass analysis of the learning objectives have then been compared with the CT definition of Grover and Pea (2013). As described earlier in this study, CT has many different definitions, but for this study, Grover and Pea's definition of CT was used in this comparison, as their definition is already mostly in the form of specific learning objectives. Through this comparison, the learning objectives associated with CT were highlighted, and an overview of the degree of CT involved in the first year of the Coderclass was given.

4.3 Student attitude outcomes

The final analysis of this study focuses on student attitude outcomes. These have been analyzed through the use of learner reports, in which the students themselves describe student attitudes.

Students of the Coderclass were asked to fill in a learning report near the end of their school year. This learning report is based on the model developed by De Groot (1980). The learner reports were filled in during class through an online survey in Google forms during the final session of the academic year. A few days before this class, students were asked to start thinking about the things they have learned in the Coderclass and things they have learned about themselves while in class so that they had a general idea what to fill in on the survey before the survey date.

The learner reports, filled in by the students, were then used to explore students' attitude outcomes towards the Coderclass. These attitudes have been explored by scanning the results of the learner reports, in which statements of the form "I have learned (or noticed, experienced) that..." are mostly encountered. The format distinguishes statements in two ways: (1) learning about 'the world' versus learning about oneself, and (2) learning of generalities (rules, things that are always the case) versus learning of exceptions (new issues, surprises). These two dimensions result in a learner report consisting of four sections. This method is copied from Barendsen and Henze (2015), who have done this in their paper for the 2015 NARST Annual International Conference, Chicago, IL. They classified statements from learner reports containing attitude aspects into the attitude components of Klop (2008) (*Cognitive, Affective, Behavioural components*) and the motivational factors of Keller (1984) (*Attention, Relevance, Confidence, Satisfaction components*) so that statements containing student attitudes could be quantified. These categories (*Relevance, Confidence, and Satisfaction*) were used as indicators for the cognitive and affective components of students' attitudes.

Cognition component:

- *Relevance.* Statements in which computer science concepts are connected to contexts. It was registered whether a connection was made to personal life or society, respectively.
- *Confindence.* Statements expressing confidence in the student's own capabilities, such as "I have noticed that I am good at ... ". Positive and negative statements were distinguished.

Affection component:

- *Satisfaction*. Statements expressing a personal preference or dislike, for example "I have discovered that I like doing ... ". Also, in this category, positive and negative formulations were distinguished.
- *Judgment*. Statements containing a judgment about a particular phenomenon, for example, "I have learned that .. is beneficial for the environment". Positive and negative judgments were distinguished.

Behavior component:

- *Behavioral intention*. Statements expressing an intention, such as "I know now that I want to do ... later".

An overview of the number of statements in each category and a more detailed overview of the findings are then presented. Furthermore, this method also summarizes the findings in terms of the quality aspects of learning outcomes, which are variation and complexity. This has also been done for this attitude analysis.

Chapter 5

Results

5.1 Conceptual content

5.1.1 Conceptual analysis

Table 5 shows the overview of the number of hits of concepts that were found during the coding in Atlas.TI of the Coderclass course materials. After that, each knowledge category is individually discussed (in alphabetical order) by naming the concepts found during coding and by discussing connections, specifics, and things that stand out.

Knowledge categories	Number of concept occurrences
Programming	137
Data	69
Algorithms	42
Architecture	41
Networking	40
Engineering	38
Modeling	33
Security	16
Mathematics	11
Usability	10
Society	8
Intelligence	1
(Total:)	445

Table 5: Number of hits of concepts found during coding related to knowledge category in course materials, sorted from most to least occurrences.

Algorithms

Concepts found: Algorithm, algorithm representation, condition, decomposition, input/output, instruction, instruction sequence, instruction set, iteration, optimization algorithm, pattern, problem-solving, redirection, redundancy, repetition, search algorithm, selection, sequence, step-form algorithm, steps.

Finding concepts in the algorithms knowledge category was mainly done through defining algorithms as a set of instructions for achieving goals, made up of pre-defined steps, as well as problem-solving using an algorithm. Algorithms is also closely related to programming. Principles of problem solving that are not connected to any specific programming language have thus been counted as an algorithms concept, while constructions in a programming language have been counted as a Programming concept. Many of the occurrences appear in an introduction to patterns in Python-0 module and in the Blender-0 module where students learn to work with Blender by making specific items through a step-form algorithm, for example,
"First discover the repetitive pattern. Then write a program that draws this pattern. Then think about reorienting the turtle so that it can write the pattern a second time. Then write a loop to draw the entire picture using repetition" (repetition, Python-0 module).
"Building a snowman; step by step tutorial" (steps, Blender-0 module).
The other occurrences are spread out throughout all of the curriculum material, as many modules include a degree of algorithms in their content. A lot of this is focussed around input/output to specific issues, such as *"This time you do not want the output of a command on the computer screen, but in a file. You can do this with something called a redirection: you tell Linux that the output doesn't go to the computer screen but to a certain file"* (input/output & redirection, Raspberry PI-0 module). These have only been counted when there was a clear algorithmic context to prevent a distorted final tally.

Architecture

Concepts found: API architecture, communication architecture, console, coordinate architecture, CPU, emulator, enigma coding machine, file IO, hardware, Linux architecture, operating system, organization structure, packet manager, raspberry PI, raspberry PI architecture, responsive design, SD-storage device, software interface, software package, storage structure, system controlling, windows architecture.

Most of the concepts found in the architecture knowledge category originate

from the Raspberry PI module in which operating systems are discussed, with a more in-depth theory about Linux, such as the following statements: *"While using the console, you are always on a certain map. The location of where you are right now is shown (in Linux) in the prompt rule, just before the $ and right after the :"* (console, Raspberry PI-0 module).
"Much used software in Linux is usually bundled in what we call packages. A package contains an entire program. You use apt-get to update the system, but also to install software. This is done through the command sudo apt install" (software package, Raspberry PI-0 module).
There are also several bits of theory about the architecture of APIs and organizational structures: *"A different way of looking at a hierarchic structure is in the form of a tree. The structure of an organization or family is often shown in such a way."* (organization structure, HTML CSS-0 module).

Data

Concepts found: Array, compression, data representation, data scientist, data sorting, data storage, data structure, data type, data visualization, file format, information, information exchange, information manipulation, information retrieval, inheritance, integer, list, meta-data, object, string, table, text representation.

The concepts in this category cover working with data and the learning about data-related concepts. There have been many occurrences of some concepts that have been omitted from the final tally, as they did not specifically cover the management of data or structures of it (e.g. "here we have some data" was omitted and "use this data to create..." was included in the tally). Furthermore, many instances of a concept have been omitted if nothing new is told about them (e.g. "An integer is a form of number..." was included while "here is another integer which we discussed earlier" was omitted). Many of the occurrences of concepts for data are located in the Raspberry PI module, where students are being taught how to manipulate data and files in Linux (especially regarding the storage of data): *"The command file gives information on a file. In Windows, every file has an extension to show what kind of file it is (like .doc for a word document, .jpg for jpeg picture, etc.). In Linux, files can also have extensions, but not necessarily. With the file command, you can thus discover what a file is."* (data representation, Raspberry PI-0 module). There are also many Data concepts in the Python-1 module where students are taught how to use different types of data in their programming: *"Mind the use of the function int()! Int is short for integer. It is the English name for a whole number"* (integer, Python-1 module). Furthermore, there are a lot of data concept hits in the Data visualization

module, as this module is mostly about data manipulation: *"You might have noticed that lists look a lot like strings. That is correct because lists and strings do share a lot of common traits: they are both row types in Python. An important difference is that the individual characters in strings cannot be changed."* (list, Data Visualization in Python-0 module) and finally in the practical assignments where students are required to visualize/manipulate data into something the client wants.

Engineering

Concepts found: Requirement, project, project management, collaboration, collaboration tools, development tools, flowcharting, present products, responsive designing, SCRUM development, data analysis (engineering), idea pitching, problem exploration, testing, web development.

Some of the concepts found in the engineering knowledge category are found in the regular course materials for different modules, covering some requirements of a specific concept or introducing other engineering concepts such as responsive designing: *"A second part of the answer is the usage of certain design rules, such as responsive design. With responsive design, we mean that an HTML-document is designed in such a way that it works well in every manner of devices, such as mobile phones or a desktop."* (responsive designing, HTML CSS-0 module).
However, most engineering concepts are found inside the group projects, in which the students use requirements given by the client to develop something using SCRUM (and thus get practical experience with engineering): *"But now you have to make a website together. That requires teamwork. Because before you know it there are multiple versions of the same page. And how do you solve that problem? You often need to merge, but that is a time consuming job. With cloud0 you can work easily on a project with multiple people"* (collaboration tools, Project 1: designing and building a website for a client),
"Below is a table of requirements which your animation must fulfill. Study this list well! In the third column you can see how important the requirement is" (requirements, Project 2: 3D animations in Blender).

Intelligence

Concepts found: Robots.

There is only a single occurrence regarding robotics in the core curriculum, found in the HTML CSS-1 module: *"The above CSS is coupled to a piece of text on Robots. Below is a screenshot of the page of designer tools on robots that is turned on"* (robots, HTML CSS-1 module).

Mathematics

Concepts found: Booleans, logical expressions (AND, OR, NOT operators).

Several modules include theory about tree structures, but often in an architectural context and are thus not counted for mathematics. The few concept occurrences in this knowledge category are introductions and applications on logical operators and booleans. There are more hits on booleans, but they are all in a programming context and offer no deeper insight into the mathematical context of the concept. These concepts are primarily found in Data visualization in Python-0 module and the Python-1 module: *"You can even tell the program that you want all numbers between 30 and 50. This is done with the & command (meaning AND)."* (AND operator, Data visualization in Python-0 module,
"Earlier at the if-expressions we have used comparisons. If this is true, then we do that! These expressions are also called boolean expressions. A boolean expression is a comparison that can only determine whether it is true or not. It cannot be maybe true or maybe untrue. Here we use the English worts of true and false." (booleans, Python-1 module),
"We can also use an or operator. With this, we check whether one of the two boolean expressions are true. If one of the two gives back true, then the whole is also true. This way, we can see a big difference between the and- and or-expressions." (OR operator, Python-1 module).

Modeling

Concepts found: (3D) animation, data visualization (graphs, piecharts), box model, client-server model (modeling), document object model, flowchart.

The major part of the concepts in the modeling knowledge category is found in 2 specific modules. The first is the Blender 3D animation/visualization module and the 3D animation project coupled to it. To prevent a warped image of the concepts occurring several times during the same tutorial, each separate Blender tutorial (in which students practice a new thing about 3D modeling) has been counted as a single occurrence: *"Building a rocket: In*

this tutorial we will create a rocket through use of the edit tool and other known tools" (3D animation, Blender-0 module).
The second module is the data visualization course in Python. Most of the modeling occurrences relate to visualizing (modeling) data: *"A different module we will be using is called Matplotlib. Matplotlib is a Python module that will be used for plotting graphs. With Matplotlib, we can easily make graphs out of data. These can be staff graphs, line graphs, histograms, etc."* (data visualization (graphs), Data visualization in Python-0 module).
There are also a few more hits in the project modules as well as the Javascript and Python modules: *"Visualizing bicycle theft in Rotterdam: create insight in data and advise your client using data visualization."* (data visualization, Project 3: Visualize the Bicycle theft).

Networking

Concepts found: Networking, URL, Client-Server Model (Networking), communication between machines, IP address, protocol, server, server interaction, browser, search engine optimization, browser standard, domain name, hyperlink, internet service, network, network address translation (NAT), packet, port forwarding, search engine, server hosting, web address, web hosting, web page, web standard, HTTP-request.

Despite the huge amount of concept occurrences of concepts such as link, website, and page, only those that have a direct connection to teaching about the concept of networking and communication have been counted as concept hits for the networking knowledge category. There are many of such occurrences in the HTML module, as this is web-based programming and many concepts such as protocols and server communication are introduced here, for example, *"The first part of the answer is standardization. HTML is the standard for the web. The different browsers try to hold up to this standard as good as possible. The current standard is HTML5; this is a living standard: new developments are added to this in steps. This works well in combination with the current evergreen browsers: these browsers update themselves regularly, with minimal trouble for the user. This means that your browser can probably do much more in a year, without you having to do much for it."* (browser standard, HTML CSS-0 module).
In the Raspberry PI module, a lot of new occurrences on web servers and IP addresses are taught, mostly regarding the use of Linux: *"On a Linux system you can easily host your own website! Your website is reachable through the internet, and everyone can see it if you wish. To do this, we will learn about the client-server model, which is the basis for all communication on the internet."* (client-server model (networking), Raspberry PI-0 module).

There are also several networking concept occurrences in the projects, particularly project 4: SMART lighting: *"The requesting of websites is done through so-called HTTP-requests through the HTTP (HyperText Transfer Protocol) protocol. Your browser (client) asks the website (server) for certain information. There are many different HTTP-requests. Today, we only need 'GET' and 'Post'. HTTP-GET asks data of the server and HTTP-POST sends data so that the server can process it."* (HTTP-request, Project 4: SMART lighting).
In general, more is being taught about the practical use of networking concepts than the theoretical background of them.

Programming

Concepts found: Python Programming, CSS coding, HTML coding, JavaScript, command, API, debugging, error, function, parameter, block programming, event, interactive program, libraries, looping, nesting, programmer, readability (code), separation of concerns, source code, variables, wiki coding.

Programming gains the highest amount of concept occurrences because almost every module is connected to a form of programming. Almost every bit of theory in the modules includes one of the concepts used for this knowledge area (such as code, functions, programming languages). Many of these concepts focus on specific programming concepts in a specific language, mostly Python, and a lot of HTML concepts, for example: *"You can see an html-element as a container with a special meaning. The tag can be used as a label on which you can write anything about the contents. You can also use it as a handle, for actions on the element. You can supply the tag with attributes that describe what happens with the contents. You can use attributes to shape the contents, like the font or color of the text."* (HTML programming, HTML CSS-0 module),
"A command is an instruction that a computer can understand and execute. A computer can only understand simple commands. These small, simple commands can then be combined to perform more complex assignments. Such a collection of commands is also called a computer program. Writing such a program is not easy. Some of these programs contain over a million lines of commands." (commands, Python-0 module),
"For-loops in JavaScript: We will briefly explain the notation of a for-loop in JavaScript below. The notation is the same as in other languages such as Java and C." (JavaScript, JavaScript for web-0 module).
Therefore it is safe to say the Coderclass curriculum has a high focus on the programming knowledge area.

Security

Concepts found: System owner/administrator, access rights, administrative rights, login/password, protection, firewall, security protocol.

Most of the security concepts are found in the modules given about Linux and the access rights/system admin of the OS: *"During the installing of Linux you created your own user account. If done right you are now logged in with that account. A Linux system also has another user. This user is called root. The root user is the administrator of the system and is allowed to do anything. It is the counterpart of the Administrator in Windows. If things must be changed in the system, then the root user has to do this. The other users can only use the system. But there is also a possibility to temporarily get the rights of the root as a normal user."* (system administrator, Raspberry PI-0 module),
"Because there are multiple users on a system, agreements on what everyone is allowed must be made. We have already seen that not just anyone can add more users or programs. You need administrative rights (superuser)." (administrative rights, Raspberry PI-0 module).
There are also a couple hits about password protection and security in general in the Projects, particularly in project 4: SMART lighting: *"As you know, the firewall is a part of the security of a computer(architecture). It keeps unwanted internet traffic out. An API is used to pass on, change, or take information without it being blocked by a firewall."* (firewall, Project 4: SMART lighting).

Society

Concepts found: Professional (software), open-source software, digital rights, ownership, hacking, social media.

The few concepts found in the society knowledge area are mostly found in the practical assignments or the Blender-0 and Raspberry PI-0 modules: *"You will work with software that is used in daily life by professionals to make films, commercials, and games."* (professional software, Blender-0 module).
"As you probably have noticed, you cannot just get into your own network through the internet. This is great because hackers and people with malicious intent cannot get in easily either!" (hackers, Raspberry PI-0 module),
In the practical assignments there is usually an explanation on why you

could benefit from a certain skill in your career or in a professional manner, or how something can be made more beneficiary for the society: *"Another possibility of making your website more accessible is through means of social media like Facebook, Twitter, LinkedIn, etc. You can adapt your website for this too.* (social media, Project 1: Design and build a website for a client). There are also a couple of hits on hacking and open source software throughout the different modules.

Usability

Concepts found: User-friendliness, efficiency, adaptability, beauty, human-computer interaction.

There is a rather large amount of references to the interaction of the user with a program, but usually in a programming or engineering perspective. In this analysis, only the concept hits regarding usability in the form of the user-friendliness, adaptability, and efficiency have been coded as usability. These hits occur in the modules of HTML-CSS-0, Raspberry PI-0, HTML-CSS-1, Python-2 and Projects 3 & 4: *"An operating system is a program that makes sure a user can work with hardware relative ease and efficiency. Operating systems have a graphical interface and a textual interface."* (efficiency, Raspberry PI-0 module),
"When the internet became more popular, the need for more beautiful websites also rose. HTML was not made for that, and so people made an expansion to HTML to make more beautiful websites. This solution is called stylesheets." (beauty, HTML CSS-1 module,)
"Typical of these smart initiatives is the innovative manner it uses ICT, through which everything gets even more efficient, better, durable and secure." (efficiency, Project 4: SMART lighting).

5.1.2 Comparing the Coderclass to other CS curricula

The number of concept occurrences in each concept knowledge category (as listed in section 5.1, table 5) can now be used as a conceptual oversight of the Coderclass curriculum. Furthermore, it can be used to compare the Coderclass to other CS curricula, to see in what ways they are alike or differ from one another, according to the same style as the curricula comparison that was done by Barendsen and Steenvoorden (2016).

Results are presented in two ways. Firstly, the categories for each curriculum, sorted according to (absolute) number of concept occurrences are listed in table 6. Secondly, the (relative) distribution of concepts across the categories for every document is shown in Fig. 4. The Dutch curriculum consists of a core curriculum and several elective themes. Therefore, the core curriculum and the curriculum as a whole (including the elective themes) are distinguished.

The total number of concept occurrences (i.e., coded quotations) is given at the bottom of each list in Table 6. The reason that France and the Netherlands have less coded concepts is that the learning objectives are formulated in a relatively small way, and it often happens that concepts are mentioned only once. The CAS and the CSTA documents formulate their guidelines in a more spiral-like way, first formulating learning objectives for lower grades and after that for higher grades. Figure 4 provides a global overview of the five documents and how they compare on the twelve respective knowledge categories and a rest category. Some of the more interesting differences between the Coderclass and the other CS curricula will then be discussed for each knowledge area.

Furthermore, it must be noted that the conceptual analysis of the Coderclass has been an analysis of complete course documents for only the first year, while the other ones have been analyses of curricula documents (guidelines rather than actual learning material) of a complete curriculum. For this reason, the Coderclass has much more total concept occurrences than the other curricula. This fact makes it hard to compare the Coderclass with the other curricula accurately. However, for the purpose of this thesis, they have still been compared, because this provides a decent overview of what the Coderclass focuses on in their first year, especially on what their strengths and weaknesses are in specific knowledge categories, as opposed to different curricula.

Coderclass Year 1

1. **Programming (137)**
2. **Data (69)**
3. **Algorithms (42)**
4. **Architecture (41)**
5. **Networking (40)**
6. **Engineering (38)**
7. **Modeling (33)**
8. **Security (16)**
9. **Mathematics (11)**
10. **Usability (9)**
11. **Society (8)**
12. **Intelligence (1)**

(Total: 445)

CSTA

1. Algorithms (44)
2. Engineering (40)
3. Architecture (37)
4. Society (30)
5. Networking (27)
6. Programming (25)
7. Data (23)
8. Security (13)
9. Modeling (12)
10. Intelligence (11)
11. Mathematics (8)
12. Usability (2)
13. Rest (0)

(Total: 272)

CAS

1. Algorithms (44)
2. Networking (40)
3. Architecture (38)
4. Data (33)
5. Programming (19)
6. Engineering (17)
7. Mathematics (5)
8. Security (4)
9. Society (2)
10. Intelligence (1)
11. Modeling (0)
    Rest (0)
    Usability (0)

(Total: 203)

France

1. Data (28)
2. Programming (15)
3. Architecture (14)
   Networking (14)
4. Algorithms (13)
5. Mathematics (8)
6. Society (5)
7. Engineering (4)
   Modeling (4)
8. Intelligence (2)
9. Rest (1)
10. Security (0)
    Usability (0)

(Total: 108)

Netherlands 2016 (core)

1. Programming (18)
2. Engineering (17)
3. Data (11)
4. Society (10)
5. Architecture (9)
6. Security (7)
7. Algorithms (6)
8. Usability (3)
9. Networking (2)
10. Intelligence (0)
    Mathematics (0)
    Modeling (0)
    Rest (0)

(Total: 83)

Netherlands 2016 (complete)

1. Programming (22)
2. Architecture (19)
   Society (19)
3. Data (18)
   Engineering (18)
   Usability (18)
4. Security (16)
5. Algorithms (14)
6. Networking (11)
7. Modeling (7)
8. Mathematics (4)
9. Intelligence (3)
10. Rest (0)

(Total: 169)

Table 6: list of knowledge categories for each curriculum documented, including the first year of the Coderclass curriculum, sorted from most to least occurring concepts. The number of concept occurrences in each category is displayed between parentheses. The total number of concept occurrences in the document is given at the end of each list.

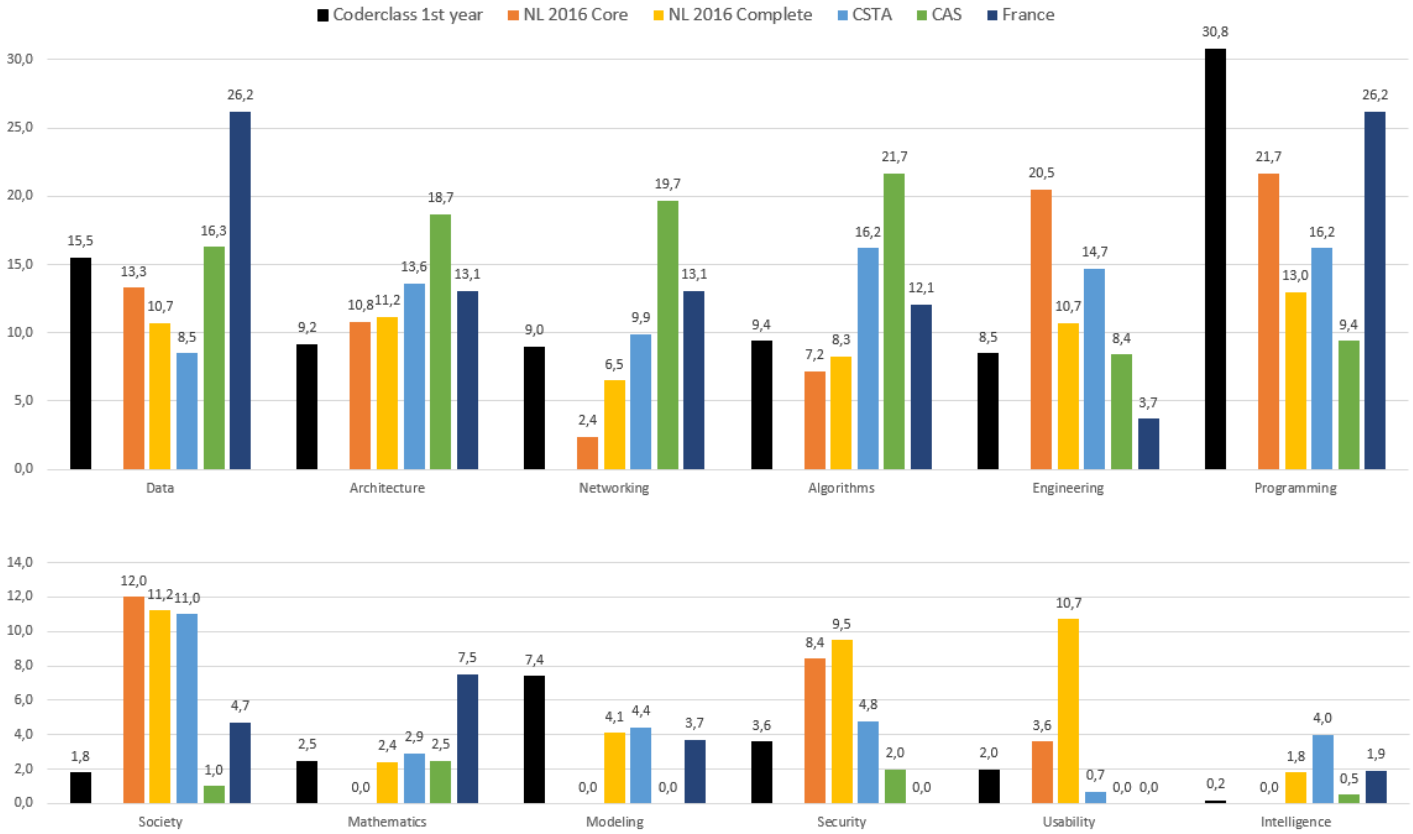

Figure 5: Relative distribution of concept occurrences across the knowledge categories. The percentages show the fraction of the concept occurrences to the respective categories. Categories are sorted according to through occurrences in Barendsen and Steenvoorden (2016), with the occurrences from the Coderclass analysis added in front.

Data

The Coderclass scores very well in comparison to the data knowledge category. Except for the French curriculum (which scores exceptionally high on this knowledge category), the fraction of data concepts involved in the Coderclass curriculum is among the highest-scoring curricula. In particular, it has a higher fraction of this knowledge category than both Dutch curricula. This is mainly because the Coderclass students work with a large amount of data and its manipulation in their first year, through the various data visualization topics and their programming with data files.

Architecture

The fraction of architecture concepts in the first year of the Coderclass curriculum, while being of moderate size, is smaller than the corresponding fractions of the other curricula. This is because there are no modules in the first year that cover computer architecture as the main topic, but architecture concepts are mentioned several times as background information in other modules.

Networking

In the networking category, the results between curricula vary. While the Coderclass has a respectable amount of concepts in this knowledge category, and the corresponding fraction scores higher than both Dutch CS curricula, it scores lower than the foreign curricula of CSTA, CAS, and France. The Dutch CS curricula generally do not include networking topics as much as foreign curricula, however, the Coderclass covers a lot of networking concepts during its first year during several net-based modules, such as the HTML module and the Raspberry PI module.

Algorithms

For the algorithms category, the same situation is present as there is for the networking knowledge category. While the Coderclass scores slightly higher than both Dutch curricula, the foreign curricula each score higher due to an increased focus on this knowledge category compared to the Dutch. The Coderclass does not feature a module in their first year with the main topic being on algorithms, yet other modules contain many algorithm concepts.

Engineering

Engineering is the first knowledge area in which the Coderclass curricula holds a significantly lower fraction of concepts than its Dutch counterparts (as well as the US-based CSTA curriculum). This is mostly because there is no specific focus on this knowledge category in any of its first-year core modules. However, the Coderclass still gains a decent fraction in this knowledge category because of the group projects students are expected to work on. In these projects, students are given an introduction to a way of cooperation that involves engineering concepts, such as working with the use of SCRUM development. Because of this, the fraction of engineering in the Coderclass is on par with its English counterpart CAS, and much higher than the French CS curricula, in which engineering has lower importance.

Programming

Programming is the main category in which the Coderclass first-year curriculum scores significantly higher than every other curriculum it has been compared to. Coderclass students start in the world of coding very early on in the course and will encounter programming concepts in almost each of the modules (of which several are mainly focussed on programming itself, such as the Python modules). Therefore the Coderclass reaches a fraction of programming concepts that are highest among its peers.

Society

The society knowledge category is underrepresented in the first year of the Coderclass, compared to the other CS curricula. Except for the English CAS curricula, it holds a lower fraction of concepts. None of the Coderclass modules have a specific focus on a societal topic, nor include a lot of secondary background material on CS in society. It particularly stands in contrast to its Dutch counterparts, which include a significantly higher focus on societal issues within CS.

Mathematics

The mathematics category has a low focus on each of the CS curricula involved in this comparison, and the Coderclass is no exception. The somewhat frequent occurrence of concepts covering logical expressions and booleans throughout the Coderclass modules ensured that its fraction of concepts on mathematics is on par with the complete Dutch CS curriculum, the CSTA curriculum, and the CAS curriculum. The French have a somewhat higher focus on mathematics in their curriculum than the others, while the core Dutch CS curriculum does not include any mathematics concepts.

Modeling

The Coderclass has a significantly higher fraction of concepts in the modeling knowledge category than the other CS curricula. This is primarily because there is a specific module in which the main category is modeling (Blender-0 module), including a similar 3D animation project. Modeling concepts are also frequently occurring in the data visualization module, which leads to a higher fraction of modeling concepts than their peers.

Security

The security knowledge category is underrepresented in the first year of the Coderclass when compared to the other Dutch CS curricula, in which security has a significant role. The English CAS curriculum also scores

slightly higher in comparison. However, when compared to the CSTA and French curricula, in which security has a low to nonexistent fraction of concepts, the Coderclass still includes a higher fraction of security concepts.

Usability

The fraction of concepts in the usability knowledge area of the first year Coderclass curriculum scores lower than its Dutch counterparts, yet the fraction is higher than the corresponding fractions of the foreign CS curricula. The complete 2016 Dutch curriculum in particular scores significantly higher in this knowledge category. The Coderclass only gains concepts in this category because of less important background information included in several modules and includes no specific usability focused module in its first year.

Intelligence

Concepts in the intelligence knowledge category are almost nonexistent for the Coderclass first-year core curriculum, apart from a single hit on robots. Thus, this category includes a lower fraction than the other CS curricula. However, the other curricula also feature a relatively low focus on this category, with the CSTA having the only outlier on intelligence.

5.2 Learning objectives

5.2.1 Learning objective analysis

The first part of this analysis is extracting earning objectives through analysis and reverse engineering of module requirements plus the assignments of each module. The extracted learning objectives are shown below for each module and are classified into core CS domains (as described by Barendsen et al. (2016)):

Coderclass-environment-0
In this first introductory module, students become acquainted with the wiki environment and Google drive in which the Coderclass operates. The module focuses mostly on learning how to use these, where to find things, and how to search for things. As such, the learning objectives in this module can be classified in the core domains of *Skills* and *Information*:

- *Skills*: students can use their information skills to look for the information they need and can communicate this back to their teachers through google drive and their personal wiki page.
- *Information*: students can identify the information they need to solve the answers to the questions given and know how to process this in their wiki/google drive.

Block-programming-0
This module serves as an introduction to programming. It allows students to become acquainted with solving puzzles through an algorithm in block programming. The learning objectives in this module can be classified in the core domains of *Skills*, *Foundations* and *Programming*:

- *Skills*: students can use their information skills to solve puzzles and can use the assigned program to develop a solution.
- *Foundations*: students can use an algorithm to solve a puzzle.
- *Programming*: students can use the program components available to them in this module to solve puzzles.

Blender-0
The Blender-0 module introduces Coderclass students to (3D) rendering and animations, through the use of the Blender tool. In this module, students create a variety of 3D models and animations. The learning objectives in this module can be classified in the core domains of *Skills*, *Information* and *Interaction*:

- *Skills*: students can use the tool Blender to model certain situations.

- *Information*: students can identify the characteristics they need to model and represent this in their creation in Blender.
- *Interaction*: students can use Blender's user interface to create elegant and well-designed models/animations and can make judge the value of their creation and the UI used in this module.

HTML-CSS-0
This module introduces students to the web-document language of HTML. The foundations of the language are used to solve problems and eventually create a website in HTML. The learning objectives in this module can be classified into the core domains of *Skills*, *Foundations*, *Information*, *Programming* and *Architecture* in some way:

- *Skills*: students can work with HTML to design and develop.
- *Foundations*: students can use the HTML specific (data) structure and grammar to solve problems and create a website.
- *Information*: students can represent data in their HTML creations in a valid HTML structure.
- *Programming*: students can use the HTML language to develop constructions and adapt/evaluate them based on the HTML code they write.
- *Architecture*: students can work with nesting and hierarchical structures in HTML.

Python-0
The Python-0 module serves as the first real programming experience for Coderclass students, in which they program a turtle to draw all sorts of figures. Students learn about many programming concepts in this module, such as loops and functions. The learning objectives in this module can be classified into the core domains of *Skills*, *Foundations* and *Programming*:

- *Skills*: students can design and experiment with the turtle to create figures and more by writing code.
- *Foundations*: students can use algorithms to create the figures they need.
- *Programming*: students can develop solutions to making figures by using the Python programming language and constructions given to them in advance. They can also inspect their code for errors and adapt/fix it if necessary.

Raspberry-PI-0
In this module, students learn about using the Raspberry PI, Linux, and

gain more knowledge about operating systems, working in a console and computer architecture in general. They also learn about several concepts regarding computer security, such as user roles and rights. The learning objectives in this module can thus be classified into the core domains of *Skills*, *Information*, *Architecture* and *Interaction*:

- *Skills*: students can use their informative skills to find specific files, and to judge or adapt them. They can also investigate the possibilities of Linux and Raspberry-PI and draw conclusions from the investigation.
- *Information*: students can work with data in the Linux OS. This includes finding, saving, editing, and structuring data.
- *Architecture*: students can explain the architecture and composition of Linux and the Raspberry PI. They can also explain the security aspects related to these architectures.
- *Interaction*: students can work with a console and more to interact with the Raspberry PI. They can also reflect on who can access certain parts of the program and files and how this relates to human interaction.

Data-visualization-in-Python-0
This module serves to further expand the Coderclass student's experience with Python through the usage of data visualization. Students learn more about data specific programming concepts, such as several types of input and output, particularly about visualizing the output in graphs. The learning objectives in this module can be classified into the core domains of *Skills*, *Foundations*, *Information* and *Programming*:

- *Skills*: students can use their informative skills and their experience gained from the Python-0 module to use and process data in their visualizations.
- *Foundations*: students can use algorithms and data structures in their assignment to create their graphs.
- *Information*: students can use specific data as input to make an adequate graphical representation as output in Python.
- *Programming*: students can use the programming language Python to develop code that will create visual representations of their input.

HTML-CSS-1
The HTML-CSS-1 module continues the HTML learning line and also introduces the Coderclass students to CSS. By using a free editor called Atom, students gain experience in using a new user interface to develop a website. Furthermore, students learn about several forms of layouts and makeup for text and pages,

such as fonts, colors, background images, borders, and more. The learning objectives in this module can be classified into the core domains of *Skills*, *Foundations*, *Information*, *Programming* and *Architecture*:

- *Skills*: students can use their skills learned from HTML-CSS-0 and the new knowledge gained about CSS to develop a website, through the use of the editor UI Atom.
- *Foundations*: students can use the specific (data) structure and grammar of HTML and CSS to solve problems and create a website.
- *Information*: students can represent data in their HTML-CSS creations in a valid HTML/CSS structure.
- *Programming*: students can use the HTML/CSS language to develop constructions and adapt/evaluate them based on the HTML/CSS code they write.
- *Architecture*: students can work with nesting and hierarchical structures in HTML/CSS.

JavaScript-for-web-0
This module is intended to broaden the Coderclass student's website building skills after having completed the HTML-CSS modules. Students learn how they can use JavaScript in combination with HTML and CSS by using the Atom editor. They learn more about CS concepts and also learn about new things that can be done by using JavaScript, which was not possible before in just HTML and CSS. The learning objectives in this module can be classified into the core domains of *Skills*, *Foundations*, *Programming* and *Architecture*.

- *Skills*: students can use their skills learned from the HTML-CSS modules and the new knowledge gained about JavaScript to develop a website through the use of the editor UI Atom.
- *Foundations*: students can use the specific (data) structure and grammar of HTML, CSS, and JavaScript to solve problems and create a website.
- *Programming*: students can combine their HTML/CSS creations with newly programmed JavaScript code and can create new web-components using the JavaScript language. students can also debug their code.
- *Architecture*: students can work with nesting and hierarchical structures in HTML/CSS and JavaScript, and can use a tree structure in their solutions.

Python-1
The Python-1 module continues on the experience gained through the Python-0 module by introducing Coderclass students to interactive programs in Python. Students learn to use user-input in their code to determine what their program does. The learning objectives in this module can be classified into the core domains of *Skills*, *Foundations* and *Programming*:

- *Skills*: students can use their skills in Python to design new interactive programs and write the code to execute them.
- *Foundations*: students can use algorithms to solve problems.
- *Programming*: students can use the Python language to create, test, and debug solutions to the assignments in this module.

Python-2
This module serves as a summary of the previous Python modules by making Coderclass students use their acquired skills to tackle a larger problem. This is done through the creating of a word game and the introduction of flowcharting. The learning objectives in this module can be classified into the core domains of *Skills*, *Foundations* and *Programming*

- *Skills*: students can use their previously acquired Python skills to design, model, and develop a word game. They can also use flowcharting and explain the way a program works.
- *Foundations*: students can use an algorithm to create a word game.
- *Programming*: students can use the Python language to create, test, and debug a word game.

Pygame-0
This module is a continuation of the Python modules in which the experience gained thus far is used to develop games. Students use the Pygame library to start their game developing and gain experience in using coordinates in an axial system to create figures and animations. The learning objectives in this module can be classified into the core domains of *Skills*, *Foundations* and *Programming*:

- *Skills*: students can use their Python skills in addition to the Pygame library to design and develop figured and animations in Python.
- *Foundations*: students can use algorithms to solve the assignments in this module.
- *Programming*: students can use the Python language and the Pygame library to develop, test, and debug solutions to the assignments in this module.

Project 1: Develop and build a website for a client
The first of the Coderclass group projects test the student's website building skills by having them develop a website for a client. Students are required to communicate with a client and determine their wishes to build an appropriate website for them. Students are also required to fulfill a list of requirements in their website design while working with planning, deadlines, and the SCRUM software development method. The learning objectives in this module can be classified into the core domains of *Skills*, *Foundations*, *Information*, *Programming* and *Interaction*:

- *Skills*: students can communicate with a client and their group to come to a website design that each party is happy with while using the knowledge gained in previous modules in a practical application. students can also develop and review the website its validity concerning the requirements.
- *Foundations*: students can use the specific (data) structure and grammar of HTML, CSS, and JavaScript to create a website for their client.
- *Information*: students can determine what their client needs and represent the client's wishes in their design while using the HTML and CSS structures.
- *Programming*: students can use the HTML/CSS. language to develop their website and adapt/evaluate it based on the feedback they gain from the client and each other.
- *Interaction*: students can reflect on the usability of the website they build, using the requirements and client wishes to make a good design.

Project 2: 3D animations in Blender
In this project, students are required to use their skills in Blender to create a bigger animation, which highlights a social issue somewhere in the world and sends a powerful message. They work with a client with additional wishes to create the animation. Furthermore, requirements have to be met in their animation, and students will again work with planning, deadlines, and SCRUM. The learning objectives in this module can be classified into the core domains of *Skills*, *Information* and *Interaction*:

- *Skills*: students can use their Blender skills and can communicate with the client and their group to come to a valid animation that adheres to client wishes and requirements.
- *Information*: students can determine what their client needs and can represent the client's wishes in their animation

- *Interaction*: students can use Blender's user interface to create an elegant and well-designed animation of their client's wishes. students can also judge the social value of a digital artifact such as a Blender animation and can see how it can be useful to send a powerful message about a socially relevant issue.

Project 3: Visualize bicycle theft
This project requires students to use their Python data visualization skills to visualize data provided to them by a client. They learn new visualization skills, such as importing excel files as input, using aggregation functions, visualizing new graphs, heat-maps, and more. Furthermore, requirements have to be met in their animation, and students will again work with planning, deadlines, and SCRUM. The learning objectives in this module can be classified into the core domains of *Skills*, *Foundations*, *Information*, *Programming* and *Interaction*:

- *Skills*: students can use their informative skills and visualization experience while working in a group to use and process client data in their project and make sure that it adheres to client wishes and requirements.
- *Foundations*: students can use algorithms and data structures to create their data visualizations.
- *Information*: students can use specific client data as input to create adequate graphical representations in a structured manner.
- *Programming*: students can write Python code that will create visual representations of their input.
- *Interaction*: students can value the elegance and usability of their visualization dashboard. students can also value a socially relevant issue by visualizing data of an important social issue.

Project 4: SMART lighting
This final project requires Coderclass students to build a website that will serve as a UI to control a Philips Hue SMART light according to a client's wishes. Students use their skills gained in the HTML-CSS modules and the JavaScript-for-web-0 module to build a dynamic website while learning about the SMART light's API, new HTML5 UI components, and client-server architectures. Furthermore, requirements have to be met on their website, and students will again work with planning, deadlines, and SCRUM. The learning objectives in this module can be classified into the core domains of *Skills*, *Programming*, *Architecture* and *Interaction*:

- *Skills*: students can work in groups and with a client, using their HTML/CSS and JavaScript skills to design and build a dynamic website to control the SMART light.

- *Programming*: students can code, design, and build a dynamic website to control the SMART light. students can also value the tidiness and elegance of their code.
- *Architecture*: students can explain the structure and function of the light's API and their website, and can explain the client-server architecture.
- *Interaction*: students can create an elegant, dynamic website to control the SMART lights with. They can also value the usability of the website.

Recurring learning objectives
Now that learning objectives in each module have been analyzed, it is possible to come to a more general summary of recurring learning objectives for the entire first year of the Coderclass. The second learning objective analysis performed reviews the entirety of the first year of the Coderclass, rather than looking at each module's specific learning objectives. When viewing the entire first year of the Coderclass in a more general approach, it becomes clear that several general objectives often occur throughout the curriculum. These general learning objectives are the following:

Cooperation: Coderclass students can work collaboratively with others to solve problems and create solutions and programs. This includes creating a program or website for a client and using SCRUM to develop solutions. This skill is especially used in the four group projects.

Debugging and testing: Coderclass students can use their programming skills to test their code and determine where faults occur so that they can fix them. They can use a systematic approach to detecting, diagnosing, and correcting errors. Many of the harder tasks and assignments in the various first-year modules (including the bigger group projects) require using multiple programming skills, and students will often encounter faults and errors in running the code, which they can now fix.

Decomposing: Coderclass students can solve a problem by tackling it in smaller steps. This is introduced in Block-programming-0, in which they tackle puzzles and mazes through step algorithms. It is also included in many other modules, such as animation and creating through tackling the problem step-by-step (Blender, HTML-CSS website and JavaScript and Python assignments).

Designing: Coderclass students can use their creativity to design new things. In the Coderclass, this mostly represents designing new systems or programs. Examples include designing their own introductory animation, websites, data visualization design and control page over controllable smart

lights.

Generalization: Coderclass students can determine shared characteristics in assignments, and use them to make simpler solutions by sharing these common features. The main idea of this skill is that students can control the complexity of their solutions. Many of such generalization issues occur during the Python oriented modules.

Initiative & Planning: Coderclass students can work with deadlines for their assignments and projects, and are thus able to plan their work accordingly. students can also show initiative by helping out others, taking up leadership, and by asking their teachers for more advanced information and tasks if they deem they can do more.

Modeling: Coderclass students can use their new skills to make representations of a real-world issue, system, or situation. This includes creating and animating in Blender, visualizing data through Python, creating visual games in Python, and creating informative websites through HTML-CSS. Students are also able to use given, existing models in their work and adapt them to their needs.

Understanding and writing a program: students can write code to execute a program, and they can understand and explain their own code and that of others. Students can find and correct mistakes in code when needed and can reflect on their program. This particular programming skill is especially highlighted in the first year of the Coderclass as this skill is involved in every module in some way.

Understanding general computer science concepts: Coderclass students can understand and explain the many issues and concepts learned about the world of computer science. A better overview of these concepts is given in section 5.1.

5.2.2 Comparing the Coderclass learning objectives to the 2016 Dutch CS learning objectives

After analyzing each separate module for the core CS learning objectives it is now possible to compare the results to that of the 2016 Dutch CS learning objectives by looking further into the subdomains of each core CS domain. Each of the core domains of Skills, Foundations, Information, Programming, Architecture, and Interaction is analyzed below, by specifying how often the required subdomains of the 2016 Dutch CS learning objectives are present in the Coderclass modules. Then, each subdomain is discussed based on the number of times it was coded throughout the core modules.

Domain A: Skills
The domain of skills is present in every module since every module trains a specific skill set of the Coderclass in some way. However, these skills vary, and it becomes apparent that some of the subdomains of skills are more common than others:

Subdomain	Number times subdomain is present in modules
A1: Using information skills	13
A2: Communicate	7
A3: Reflect on learning	0
A4: Orienting on profession and study	2
A5: Research	3
A6: Modelling	6
A7: Appreciate and judge	2
A8: Design and develop	13
A9: Computer science as a perspective	8
A10: Collaboration and interdisciplinarity	4
A11: Ethical discourse	1
A12: Wielding computer science instruments	14
A13: Working in context	7

Table 7: Number of times each subdomain of domain A: Skills is present throughout the Coderclass modules.

- *A1: Using information skills*: This subdomain has a high presence in the modules and is thus well represented throughout many of the modules.
- *A2: Communicate*: This subdomain is represented mostly through the projects in which teams of students have to work together and

communicate to come to a joint solution to a problem.

- *A3: Reflect on learning*: This particular skill is not specifically focussed on in any of the modules.
- *A4: Orienting on profession and study*: This skill is not particularly addressed in the modules, but is indirectly mentioned in some of the modules and projects, in a way that lets the Coderclass students reflect on what the things they are doing and learning can be used for in professional life.
- *A5: Research*: Proper researching skills are not yet tested much throughout the first year. Some of the modules do require students to do a bit of research, but not yet at the required level.
- *A6: Modeling*: Modeling is trained in several of the modules, in which students have to analyze a relevant issue and translate it to one they model, test, and judge.
- *A7: Appreciate and judge*: This skill does not specifically feature much in the Coderclass modules, as there are not many assignments in which students have to judge a situation or application in a scientific or societal manner. However, some personal arguments in judgement are made throughout some of the modules.
- *A8: Design and develop*: This skill has a high presence in the Coderclass modules. By use of many digital artifacts, students have to design and develop all kinds of solutions to problems and execute them throughout many of the modules and projects.
- *A9: Computer science as a perspective*: The first year of the Coderclass features a high presence of chances for students to use CS as a perspective by teaching students about CS specific terms, concepts and relations.
- *A10: Collaboration and interdisciplinarity*: This skill is trained mostly during the group projects, but is not particularly mentioned in the other modules.
- *A11: Ethical discourse*: This skill is not particularly trained in any of the modules in the first year of the Coderclass and only has one occurrence.
- *A12: Wielding computer science instruments*: Because students are expected to use many forms of digital artifacts and CS-related tools, this subdomain reaches a high count of occurrence in the modules.
- *A13: Working in context*: Coderclass students are instructed on the value their newly acquired skills have in a social/professional context,

particularly through the group assignments.

Domain B: Foundations
The foundations domain is represented in most of the modules and projects, but not each of the subdomains is as common:

Subdomain	Number times subdomain is present in modules
B1: Algorithms	7
B2: Datastructures	5
B3: Automatons	1
B4: Grammars	4

Table 8: Number of times each subdomain of domain B: Foundations is present throughout the Coderclass modules.

- *B1: Algorithms*: This foundation is mostly trained throughout the use of algorithms in many ways, and is present in most of the Coderclass modules.
- *B2: Datastructures*: The use of data structures is also present throughout some of the Coderclass modules in which students use, compare, and judge data structures for elegance and efficiency.
- *B3: Automatons*: This foundation does not reach a higher presence in the modules because the use of algorithms in modules is usually not specifically defined with finite-state automatons.
- *B4: Grammars*: While students have to use a predefined grammar in some modules, this is not particularly addressed as such.

Domain C: Information
Learning objectives in the information domain occur more than some of the other domains, as students often need to use, identify, and represent specific information to achieve a specific goal:

- *C1: Goals*: The goals subdomain is well represented because students gain experience in using information for different goals, and are trained in looking for the right information and the editing of it.
- *C2: Identification*: This subdomain is also well represented in the first year of the Coderclass. students often have to identify the right data to solve a specific problem in many of the assignments.

Subdomain	Number times subdomain is present in modules
C1: Goals	7
C2: Identification	7
C3: Representation	8
C4: Standard representations	3
C5: Structured data	2

Table 9: Number of times each subdomain of domain C: Information is present throughout the Coderclass modules.

- *C3: Representation*: Through the use of many artifacts and dealing with data, Coderclass students gain experience in representing data in a specific structure, particularly in the data visualization modules.
- *C4: Standard representations*: Standard representations of data are made in some of the data visualization modules in which students have to relate the outcomes.
- *C5: Structured data*: While having a lower presence than the other subdomains, students gain some experience in translation a need for information into a collection of structured data.

Domain D: Programming
The domain of programming has a high presence in the Coderclass modules and projects, as was already established in section 5.1. Almost each of the modules requires students to program, develop, and work with some form of coding:

Subdomain	Number times subdomain is present in modules
D1: Develop	10
D2: Inspect and adapt	10

Table 10: Number of times each subdomain of domain D: Programming is present throughout the Coderclass modules.

- *D1: Develop*: Coderclass students gain hands-on experience in developing programs and components using imperative programming languages in many of the modules and projects. students are also somewhat trained in using abstraction and code structuring.

- *D2: Inspect and adapt*: Aside from developing, Coderclass students are also trained in being able to inspect program components and explain, evaluate and adapt them throughout their first year.

Domain E: Architecture
The architecture domain is has a lower presence than other domains in the first year of the Coderclass. While some architectures are discussed in some modules, such as the Linux OS architecture, the subdomain of Security does not contain many learning objectives:

Subdomain	Number times subdomain is present in modules
E1: Decomposition	5
E2: Security	1

Table 11: Number of times each subdomain of domain E: Architecture is present throughout the Coderclass modules.

- *E1: Decomposition*: In some of the modules, Coderclass students gain experience in recognizing architectural elements and start to use them to interpret digital artifacts and their interaction.
- *E2: Security*: Architectural elements related to Security do not reach a high presence in the Coderclass modules. While students gain some knowledge of Security behind Linux, they are not trained in recognizing security threats in their first year.

Domain F: Interaction
Interaction does have some importance in the first year's modules but focuses mostly on usability. These interaction aspects do appear particularly in projects where usability and elegance are of importance:

- *F1: Usability*: Usability has a higher presence than the other interaction subdomains in the first year Coderclass modules and assignments. students gain experience in using, evaluating, and designing interfaces and judging them on usability and elegance.
- *F2: Social aspects*: Social aspects occur a few times in some of the projects, where Coderclass students can see how a certain piece of data or a program can impact or help their society and themselves.
- *F3: Privacy*: This subdomain is not explicitly present in any of the first year Coderclass modules and projects.

Subdomain	Number times subdomain is present in modules
F1: Usability	6
F2: Social aspects	2
F3: Privacy	0
F4: Security	1

Table 12: Number of times each subdomain of domain F: Interaction is present throughout the Coderclass modules.

- *F4: Security*: Interaction elements related to Security have a single occurrence in the Coderclass modules.

Summary
When summarizing the overview of the comparison between the first year Coderclass modules/projects and the subdomains of the 2016 CS core exam program given above, it becomes apparent that the core domains of *Interaction* and *Architecture* are somewhat underrepresented in the first year of the Coderclass. However, when it comes to the core domain of *Programming* it becomes clear that students start to use the skills described in the subdomains often and should have no problem picking up the learning objectives related to this domain. Furthermore, apart from subdomains B3 and C5, the domains of *Information* and *Foundations* also have a high presence. When it comes to the main domain of *Skills*, the first year of the Coderclass focuses mostly on some of the subdomains (A1, A2, A6, A8, A9, A12, A13) and less so on others (A3, A4, A5, A7, A10, A11).

5.2.3 Computational thinking in the Coderclass

The set of general Coderclass learning objectives found through the secondary analysis in section 5.2.1 can then be compared to one of the more broadly agreed on definitions of CT. For this study, the definition of CT, as defined by Grover and Pea (2013), is used in the comparison between CT skills and Coderclass learning objectives. By comparing the Grover and Pea definition of CT with the Coderclass learning objectives found in the second part of section 5.2.1, the degree of CT that arrises within the learning objectives of the Coderclass can be analyzed and discussed.

In this comparison, it becomes apparent that 5 out of 9 of the overall Coderclass learning objectives found in 5.2.1 have much in common with the definitional list of skills defined as CT by Grover and Pea. Table 13 displays each of the Grover and pea CT definitions and shows which of the Coderclass learning objectives are relevant to that specific definition:

Grover and Pea CT definitions	**Relevant Coderclass learning objectives**
Abstractions and pattern generalizations (including models and simulations	Decomposing, Generalization, Modeling
Systematic processing of information	Generalization
Symbol systems and representations	Modeling
Algorithmic notions of flow of control	Decomposing
Structured problem decomposition (modularizing)	Decomposing
Iterative, recursive, and parallel thinking	Generalization, Understanding and writing a program
Conditional logic	Understanding and writing a program
Efficiency and performance constraints	Understanding and writing a program
Debugging and systematic error detection	Debugging and testing

Table 13: CT definitions by Grover and Pea and the overall Coderclass learning objectives they aline with.

Each of the Grover and Pea definitions of CT alines to some extent with at least one of the learning objectives that have been found for the first year of the Coderclass. Therefore, students of the Coderclass should start to be able to think like a computer scientist over time and will be able to use CT to solve problems.

However, not all of these CT skills are represented as heavily as the others. While most of the Gover and Pea CT skills closely are related to programming and modeling (such as debugging and systematic error detection, iterative, recursive and parallel thinking, symbol systems and representations etc) and will thus stand out considering the heavy focus on these areas (as discovered in section 5.1), some of the CT skills that focus more on other areas like mathematics or usability are not as well represented. These are conditional logic and efficiency and performance constraints in particular.

5.3 Student attitude outcomes

The learner reports used to analyze the student attitude outcomes were filled in by a total of 20 Coderclass students and contain a total of 154 statements (on average 7,7 statements per student) of which 105 contained aspects of attitude components. The overall result of the classification of the statements made by Coderclass students is displayed in table 14.

Attitude Component (Klop, 2008)	Category (cf. Keller, 1984)	Specification	Number of statements
Cognition	Relevance	personal life	2
		society	5
	Confidence	positive	38
		negative	11
Affection	Satisfaction	positive	24
		negative	3
	Judgement	positive	12
		negative	7
Behavior	Behavioral intentions		3

Table 14: Distribution of Coderclass students' statements over attitude components.

A more detailed overview of the findings of this analysis, organized by category and including a lot of examples, is presented below.

Relevance

This category includes statements that show the student's perceived relevance of the various content they have learned throughout the year in the Coderclass. There are very few such statements, as most of the students have filled in about the topics they have learned about without making any connections to the real world or their personal lives. The two statements that resemble a connection to personal life are about the students discovering that there are multiple ways of solving a problem: *"I've learned that I can use many different ways to tackle a problem"*, and that there are similarities between the programming languages that they learn about in their first year: *"I noticed many of the programming languages we used are much alike."*

Statements concerning societal relevance are slightly more frequent and often refer to students now realizing how big the world of computer science actually is: *"I've realized even animation is a part of computer science"*, *"I've noticed that there is much more to the world of computer science than I thought, like animations and robotics."*. Some of the students have also

discovered that this world of CS is so big that they will most likely never be able to learn everything: *"I now know that the world of coding is infinite"*, *"I've learned that there is too much information ever to be able to learn it all.*

Confidence

This category contains statements about students' faith in their own capabilities concerning the topics they encountered during their first year in the Coderclass. Most of these statements are positive (38 out of 49 statements). Students often state their confidence in their own capabilities concerning different specific topics being taught in the Coderclass, such as visualization in Python: *"Making a graph in Python is actually quite easy"*, building websites in HTML: *"I found out I am good in HTML"* and animations in Blender: *"Animation is much easier than I thought once I understood it"*.

There are also several positive statements about more general skills, such as programming and building websites in general: *"You see websites everywhere and now I know I can build them too!"*. There are also quite a lot of general statements in which students newfound confidence in areas they did not initially expect: *"I have managed to build things in this class I would never have thought I was capable of making"*, and some praise the Coderclass itself in their statement: *"In the first year in the Coderclass I have learned more than I have in any other IT related subject"*. The most distinguishable result here is that many of the Coderclass students expected things to be much harder than they actually were, and often underestimated their own capabilities.

However, there have also been some negative statements (11 out of 49), in which the opposite was mostly the case, either expected: *"Though I did expect this class would be very hard and complicated"*, or unexpected: *"Things were more difficult than I thought they would be"*. Several statements on a student's confidence are about more specific concepts, such as the difficulty of Python: *"Programming in Python is hard for me"* or reaching a deadline: *"I've learned that I am not that good in reaching a deadline"*.

Satisfaction

In this category, statements concerning the way students feel about what they have learned are included. These are almost all positive (24 out of 27). For the most part, these statements express the students' joy in specific concepts that occur during the first year of the Coderclass, particularly about Blender: *"Modeling in Blender was great"*, Javascript: *"Javascript was a lot of fun, more so than the other topics"*, and Python: *"Making*

games in Python was the most fun I've had in the Coderclass"

Some of the statements discuss a more general topic such as the Coderclass itself: *"5 hours of Coderclass a week is a lot of fun, because you have much time to work on the modules"*, or about programming is much more fun than the students thought it would be: *"I found out programming is much more fun than I thought"*, *"I did not expect coding to be this much fun"*.

There are also three statements that express a more negative attitude. One of them expresses a student's wish for change in the Coderclass: *"It would be nicer and more fun if we could tackle the modules with our own ideas some more"*, and two of the statements state the students' negative experience with Raspberry-PI, which are the only Raspberry PI related statements in this analysis: *"I didn't like Raspberry-PIs"*, *"I noticed Raspberry-PIs aren't as much fun."*

Judgement

This category contains direct judgement made by students. Most of these statements are quite positive (12 out of 19 judgement statements are positive). Only a few of these cover a specific concept or digital artifact, such as *"I learned that Trello is a very nifty program"*, as most of the judgements made by students are about a specific concept being taught in the Coderclass, but are mostly about the Coderclass course as a whole, covering many different aspects of the Coderclass, such as its creativity: *"I've learned that the Coderclass is very creative"*, and praising of the teachers: *"The teachers of the Coderclass are very good in what they do and are very creative with their assignments and projects"*.

Some of the judgements made by the students give a more negative view (7 out of 19). However, two of those are judgements made on the time they have available for the Coderclass each week and seem to indicate that the students wish they had more time: *"5 hours a week is too little time"*, *"5 hours a week is too little. And because of (other) homework, I can't always do work for the Coderclass as much as I would want"*. The other negative judgements are about students reconsidering if the Coderclass or CS in general is really what they initially thought: *"I thought computer science would be perfect for me, but that ended up not being true"*, *"The Coderclass is different than I expected because we did more programming than I thought and did fewer things like editing and making games"*. Finally, one of the judgements reflects a student's newfound view on collaboration: *"Collaborating with others is rather hard sometimes"*.

Behavioral intentions

This category includes statements expression intentions. This is the smallest category with only three statements as students did not express much of their intentions in their statements. Two of these express intentions of inquisitive students that have found out new intentions: *"I like using a (programming) language to make something it wasn't really intended for," "I found out I like to move on with different topics before I finished the last one"*. The third statement reflects a student's intention to ask more questions in class: *"I need to ask more questions; otherwise, I will never solve the problems I face in class"*.

Chapter 6

Conclusion

Conceptual content

1. **What is the conceptual content of the first year of the Coderclass?**

   The conceptual content of the first year of the Coderclass consists of a large list of concepts that appear in the curriculum. These concepts are divided into knowledge categories to depict which categories the Coderclass focuses on most, and which categories have a lower amount of focus during the first year:

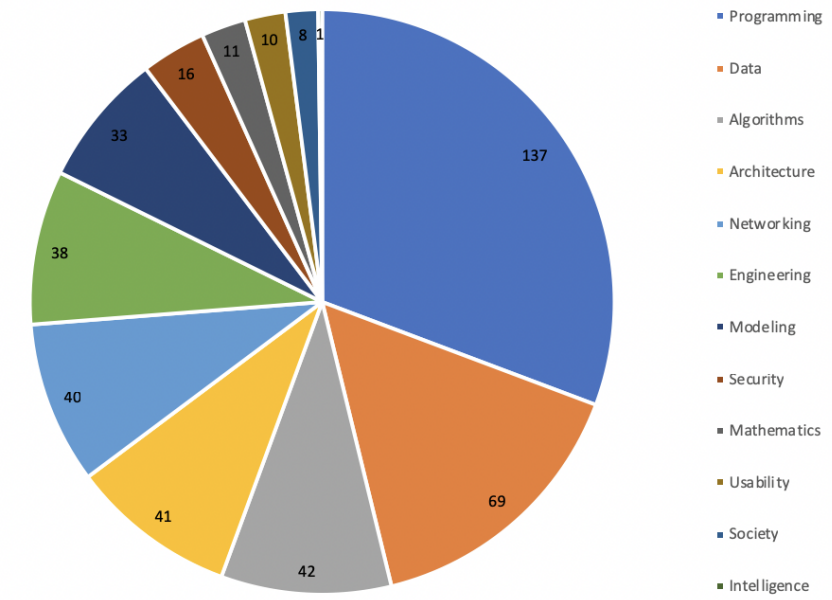

Figure 6: Concept piechart of displaying the number of concepts found in each knowledge category.

Figure 6 displays the distribution of the concepts found among knowledge categories. Programming particularly stands out as it contains the highest amount of concepts. The data knowledge category also contains significantly more concepts than other categories. Security, mathematics, usability, society, and intelligence contain a relatively low amount of concepts.

2. **How does the content of the Coderclass compare to other curricula?**

   The first year of the Coderclass covers content that is more densely distributed among certain knowledge categories than others. When comparing it to several other CS curricula (NL 2016 Core & Complete curricula, CSTA, CAS, and France), it is clear that the Coderclass scores much better than others in the knowledge categories of *Programming* and *Modeling*. It also scores above average in the *Data* and *Networking* categories. On the knowledge categories of *Algorithms* and *Mathematics*, the Coderclass scores averagely well in comparison. The *Architecture*, *Engineering*, *Security*, *Usability* and *Intelligence* categories are a bit underrepresented compared to the other CS curricula. The *Society* knowledge category scores particularly low for the Coderclass in comparison.

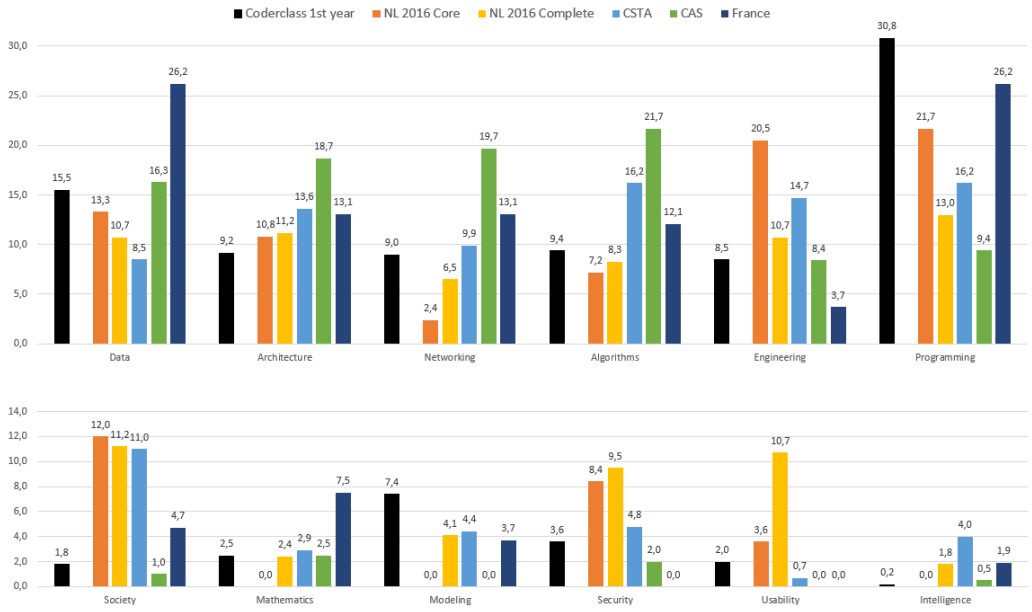

Figure 7: Relative distribution of concept occurrences across the knowledge categories. The percentages show the fraction of the concept occurrences to the respective categories.

Learning objectives

3. **What are the learning objectives in the first year of the Coderclass?**

   The recurring learning objectives that are present throughout the first year of the Coderclass are: cooperation, debugging and testing, decomposing, designing, generalization, initiative & planning, modeling, understanding and writing a program, understanding general computer science concepts.

4. **How do the learning objectives of the Coderclass compare to the Dutch Computer Science learning objectives?**

   In the comparison between the core Dutch CS 2016 Computer Science learning objective and the Coderclass first-year learning objectives, it becomes apparent that the core domains of *Interaction* and *Architecture* are somewhat underrepresented in the first year of the Coderclass. However, the core domain of *Programming* is very present, and it quickly becomes clear that students start to use the skills described in its subdomains very often in their first year and should have no problem picking up the learning objectives related to this domain. Furthermore, the domains of *Information* and *Foundations* are also well represented. When it comes to the main domain of *Skills*, the first year of the Coderclass focuses mostly on some of the subdomains and less so on others.

5. **In what way do the learning objectives of the Coderclass involve Computational Thinking?**

   The set of recurring learning objectives found in most of the Coderclass first-year curriculum have much in common with one of the more broadly agreed on definitions of CT by Grover and Pea (2013). 5 out of 9 of the Coderclass learning objectives (Debugging and testing, decomposing, generalization, modeling, understanding and writing a program) cover each of Grover and Pea's definitions of CT to some extent. Therefore, students of the Coderclass should start to be able to think like a computer scientist over time and will be able to use CT to solve problems.

Attitude outcomes

6. **Which learning outcomes can be determined concerning Coderclass student attitudes?**

   Outcomes regarding student confidence, satisfaction, and judgement on the Coderclass are particularly positive about their first year in the Coderclass. They express their joy, confidence in their own skill and positive judgement in both specific modules and in general CS concepts. The relevance of learned content and the intent to act upon them is underrepresented in Coderclass student attitudes:

Category	*student's attitude outcomes*
Relevance	Very few statements on personal life. Little more statements on societal aspects but limited variation and conceptually weak.
Confidence	Varied on conceptual content, but mostly positive statements. students express their newfound confidence on specific topics and general skills, particularly on how it was easier than students expected. The fewer negative statements are quite varied on concepts.
Satisfaction	Particularly positive; statements express how much 'fun' was had in specific modules of the Coderclass curriculum. Various statements also express joy in more general CS aspects such as programming. The few negative statements all refer to one specific module.
Judgement	Varied but mostly positive, express judgement about the Coderclass course as a whole.
Behavioral intent	Present in just a few statements, express various intentions about things that have been found out during the course.

Table 15: Summary of findings w.r.t. attitude outcomes per category

Chapter 7

Enriching the Coderclass

Up until this chapter, this thesis has been about studying the Coderclass. However, this chapter is about helping it develop, by contributing a new module for Coderclass students to study.

Before starting the Coderclass study, an arrangement with the Coderclass teachers was made. While I would be able to study everything I needed and ask anything, I would have to repay the favor by developing a new module. By looking at the conceptual analysis it becomes clear that some of the knowledge areas (particularly society, security, usability and intelligence) are underrepresented in the first year of the curriculum. Because of this I wanted to do something with one of these knowledge categories. While some of these knowledge categories are picked up more over the next couple years of the Coderclass, the society category seems to stay at a low percentage of relative distribution of concept occurrences. Because of that, I have chosen to create a module that focuses more on the society category.

The next step was to determine the subject of the module. After suggesting some initial ideas that I developed based on my own experiences of my Information Science education at Radboud university, the Coderclass teachers and I agreed on a module that focuses on innovation and entrepreneurship. The main question to be answered being:"How can I turn a revolutionary idea (specifically involving IT) into a business?". This module would be the beginning of a completely new entrepreneurship in IT learning line in the Coderclass. While this line is optional, it gives Coderclass students the option of developing an entrepreneurial mind in an IT context.

This module idea is mostly based on one of my own educational experiences, in which the same idea is applied (albeit in a much broader sense) in a more advanced setting. In the Software Development Entrepreneurship course at Radboud University Nijmegen, I had to come up with and develop my own

business idea, which ended with a sales pitch in front of a panel of experts. This experience introduced me to entrepreneurship in a rather fun way, so the idea was to give Coderclass students a light version of this experience.

The forming and design of this module (of which the front page can be seen in Dutch in Appendix 9.2) came together by copying the design of other existing Coderclass modules, while adding an introduction on innovation and entrepreneurship, including several questions for the students to reflect on. However, the main body of the module is for the students to be introduced to the concepts of entrepreneurship and innovation in IT through means of inspiration. students are asked to reflect on current revolutionary IT ideas after which they will have to interview some IT startups near them. These interviews are meant to inspire Coderclass students and to come up with a new IT business idea themselves. This idea is a start of the entrepreneurial thinking that the students will be developing during the entrepreneurship modules. They will have to think about many aspects such as the way they will be making profits, competition, etcetera. This will all have to be pitched in a sales talk in front of the class to see how well they have done working out their ideas, and students will finally be judged on their sales pitch by their teachers.

To prepare the specifics of this module, I have used a PCK model on lesson preparation form to help design the setup of the module. PCK stands for Pedagogical Content Knowledge, a term introduced by Shulman (1986) and described by him as "that special amalgam of content and pedagogy that is uniquely the province of teachers, their own special form of professional understanding" (Shulman, 1987, p. 8). This combination allows teachers to help students understand complex concepts. The PCK lesson preparation model used to develop this module helps teachers to prepare a lesson, or even a subject, in a more methodical manner, so that the teacher will be better prepared while a student will find it easier to understand. The preparation for the Entrepreneurship-0 module according to the PCK model is described in a more systematic manner in the following section.

7.1 Module preparation according to PCK model

7.1.1 What and why

What do you want students to learn about this subject? Create specific learning objectives: "After the lesson (module), the student can ...". Why is it important that they can do/know this? (Relevance of subject matter in regard to core learning goals, and meaning to the students)

Learning objectives

After completion of the module Entrepreneurship-0, the student can:

- explain the concepts of entrepreneurship, innovation and disruption;
- link these concepts to the world of computer science;
- interview an IT startup business, and use the results to gain inspiration;
- explain the concepts of a business plan;
- create a mind-map of a business plan for a business idea involving IT;
- pitch a business idea in front of peers.

Relevance of subject matter

The added value of this module for the Coderclass is that:

- students learn about the link between computer science and business;
- students develop an entrepreneurial mind in an IT context;
- this module acts as a basis for the Entrepreneurship line in a fun and light manner;
- a module that focuses more on the society concept knowledge category is added;
- it gives students the possibility to come up with an idea they can turn into a business as part of their education. This can then also be used in other subjects, such as economics.

7.1.2 Possibilities and limitations

Study the subject matter in regard to expected prior knowledge, student ideas and thinking level. Which difficulties do you expect on the subject and the learning objectives for this lesson (module)? What other factors (such as classroom, ambiance, group orientation, independence, differences between students etc) influence your teaching on this subject?

Areas of concern/points of interest

- Students do not need any prior knowledge of entrepreneurship or business to start this module. The module is meant as a start to these concepts.
- Students do need some prior knowledge about IT related concepts, in order to come up with an idea which they will turn into a business proposal. Because of the many different areas they can use to generate an idea (such as IT in traffic, IT in gaming, etc) no specific knowledge is required.
- Students must to be open to the idea of visiting an IT business and interviewing them on their ideas, as this is part of the requirements for completing this module.
- Time must be made for the students to pitch their business idea in front of the entire class.
- Students must be able to work in groups, as the project in this module should be performed in groups of 2 to 4 students, in which they must collaborate to generate a business idea and create a business plan mind-map.
- The setup for this module generates possibilities for other subjects, such as economics.

7.1.3 Approach of education

What teaching approach (such as selection and order of the subject matter, contexts, teaching methods, educational tools, verbal/visual stimuli, representations etc) do you choose, and why? Your approach should be aimed towards realizing the learning goals, and assumes the difficulties and assumes the possibilities and limitations of the starting situation

Approach

The following tools/teaching methods are used in this module:

1. The Coderclass wiki page for the Entrepreneurship-0 module.
2. Document on google containing subject matter .
3. Document on google in which the course project is explained.
4. Module project: The interviews with IT startups.
5. Module project: The idea + business plan mind-map.
6. Module project: The pitch in front of the class.

Reason

1. The Coderclass wiki page is the start for every module. Here, students access all the other necessary documents that are linked to this module, and they can read what the module is about. As this module is an elective one, students can easily decide whether this module appeals to them by reading the summary on the subject.

2. The document with subject matter on google contains the necessary information for each student to start the module project. This is the usual manner for Coderclass modules. students must each read through this document, so that all students in a group have knowledge of the necessary concepts in the project. The theory in this document also establishes the link between IT and business, and forces the student to start thinking about ideas of their own.

3. The document on google containing the module project is necessary for students to understand what they are required to do to complete this module. It contains the requirements of the project and also includes recommendations and tips to help students on their way.

4. The interviews with IT startups are meant to inspire students. Interviewing smaller startups show students that many such businesses start with just an innovative idea, which they could potentially think of themselves. Furthermore, this approach introduces students to performing interviews.

5. The idea and business plan mind-map are the core idea behind the project. students must work together in groups to come up with a good business idea involving IT and then map out their idea into a business plan. A mind-map is used to give students the option of creating a light version of a business plan in which the most important topics are covered, so that they will not have to create an entire, full-length business plan. By doing this, students learn to work together in order to make something of a joint idea, in which they will potentially become partners. It also forces the students to use the knowledge gained from the google document in a practical assignment.

6. The pitch of the business idea/plan in front of the class is meant as the final presentation of this module. By sufficiently pitching their idea, they can convince their teachers and others that they have picked up on the concepts of this module and that their idea is (potentially) a good one. students also gain experience in presenting and winning over other students.

7.1.4 Assessment

How do you discover the prior knowledge and the learning outcomes (students either understand the subject matter or are confused by it) in regard to this subject?

This is done through the students' execution of the module project. By pitching their idea and mind-map in front of the class, added by any other documents made by the students, teachers can adequately assess whether the concept of entrepreneurship in IT has landed with the students.

Chapter 8

Enriching the Coderclass

Up until this chapter, this thesis has been about studying the Coderclass. However, this chapter is about helping it develop, by contributing a new module for Coderclass students to study.

Before starting the Coderclass study, an arrangement with the Coderclass teachers was made. While I would be able to study everything I needed and ask anything, I would have to repay the favor by developing a new module. By looking at the conceptual analysis it becomes clear that some of the knowledge areas (particularly society, security, usability and intelligence) are underrepresented in the first year of the curriculum. Because of this I wanted to do something with one of these knowledge categories. While some of these knowledge categories are picked up more over the next couple years of the Coderclass, the society category seems to stay at a low percentage of relative distribution of concept occurrences. Because of that, I have chosen to create a module that focuses more on the society category.

The next step was to determine the subject of the module. After suggesting some initial ideas that I developed based on my own experiences of my Information Science education at Radboud university, the Coderclass teachers and I agreed on a module that focuses on innovation and entrepreneurship. The main question to be answered being:"How can I turn a revolutionary idea (specifically involving IT) into a business?". This module would be the beginning of a completely new entrepreneurship in IT learning line in the Coderclass. While this line is optional, it gives Coderclass students the option of developing an entrepreneurial mind in an IT context.

This module idea is mostly based on one of my own educational experiences, in which the same idea is applied (albeit in a much broader sense) in a more advanced setting. In the Software Development Entrepreneurship course at Radboud University Nijmegen, I had to come up with and develop my own

business idea, which ended with a sales pitch in front of a panel of experts. This experience introduced me to entrepreneurship in a rather fun way, so the idea was to give Coderclass students a light version of this experience.

The forming and design of this module (of which the front page can be seen in Dutch in Appendix 9.2) came together by copying the design of other existing Coderclass modules, while adding an introduction on innovation and entrepreneurship, including several questions for the students to reflect on. However, the main body of the module is for the students to be introduced to the concepts of entrepreneurship and innovation in IT through means of inspiration. students are asked to reflect on current revolutionary IT ideas after which they will have to interview some IT startups near them. These interviews are meant to inspire Coderclass students and to come up with a new IT business idea themselves. This idea is a start of the entrepreneurial thinking that the students will be developing during the entrepreneurship modules. They will have to think about many aspects such as the way they will be making profits, competition, etcetera. This will all have to be pitched in a sales talk in front of the class to see how well they have done working out their ideas, and students will finally be judged on their sales pitch by their teachers.

To prepare the specifics of this module, I have used a PCK model on lesson preparation form to help design the setup of the module. PCK stands for Pedagogical Content Knowledge, a term introduced by Shulman (1986) and described by him as "that special amalgam of content and pedagogy that is uniquely the province of teachers, their own special form of professional understanding" (Shulman, 1987, p. 8). This combination allows teachers to help students understand complex concepts. The PCK lesson preparation model used to develop this module helps teachers to prepare a lesson, or even a subject, in a more methodical manner, so that the teacher will be better prepared while a student will find it easier to understand. The preparation for the Entrepreneurship-0 module according to the PCK model is described in a more systematic manner in the following section.

8.1 Module preparation according to PCK model

8.1.1 What and why

What do you want students to learn about this subject? Create specific learning objectives: "After the lesson (module), the student can ...". Why is it important that they can do/know this? (Relevance of subject matter in regard to core learning goals, and meaning to the students)

Learning objectives

After completion of the module Entrepreneurship-0, the student can:

- explain the concepts of entrepreneurship, innovation and disruption;
- link these concepts to the world of computer science;
- interview an IT startup business, and use the results to gain inspiration;
- explain the concepts of a business plan;
- create a mind-map of a business plan for a business idea involving IT;
- pitch a business idea in front of peers.

Relevance of subject matter

The added value of this module for the Coderclass is that:

- students learn about the link between computer science and business;
- students develop an entrepreneurial mind in an IT context;
- this module acts as a basis for the Entrepreneurship line in a fun and light manner;
- a module that focuses more on the society concept knowledge category is added;
- it gives students the possibility to come up with an idea they can turn into a business as part of their education. This can then also be used in other subjects, such as economics.

8.1.2 Possibilities and limitations

Study the subject matter in regard to expected prior knowledge, student ideas and thinking level. Which difficulties do you expect on the subject and the learning objectives for this lesson (module)? What other factors (such as classroom, ambiance, group orientation, independence, differences between students etc) influence your teaching on this subject?

Areas of concern/points of interest

- Students do not need any prior knowledge of entrepreneurship or business to start this module. The module is meant as a start to these concepts.
- Students do need some prior knowledge about IT related concepts, in order to come up with an idea which they will turn into a business proposal. Because of the many different areas they can use to generate an idea (such as IT in traffic, IT in gaming, etc) no specific knowledge is required.
- Students must to be open to the idea of visiting an IT business and interviewing them on their ideas, as this is part of the requirements for completing this module.
- Time must be made for the students to pitch their business idea in front of the entire class.
- Students must be able to work in groups, as the project in this module should be performed in groups of 2 to 4 students, in which they must collaborate to generate a business idea and create a business plan mind-map.
- The setup for this module generates possibilities for other subjects, such as economics.

8.1.3 Approach of education

What teaching approach (such as selection and order of the subject matter, contexts, teaching methods, educational tools, verbal/visual stimuli, representations etc) do you choose, and why? Your approach should be aimed towards realizing the learning goals, and assumes the difficulties and assumes the possibilities and limitations of the starting situation

Approach

The following tools/teaching methods are used in this module:

1. The Coderclass wiki page for the Entrepreneurship-0 module.
2. Document on google containing subject matter .
3. Document on google in which the course project is explained.
4. Module project: The interviews with IT startups.
5. Module project: The idea + business plan mind-map.
6. Module project: The pitch in front of the class.

Reason

1. The Coderclass wiki page is the start for every module. Here, students access all the other necessary documents that are linked to this module, and they can read what the module is about. As this module is an elective one, students can easily decide whether this module appeals to them by reading the summary on the subject.

2. The document with subject matter on google contains the necessary information for each student to start the module project. This is the usual manner for Coderclass modules. students must each read through this document, so that all students in a group have knowledge of the necessary concepts in the project. The theory in this document also establishes the link between IT and business, and forces the student to start thinking about ideas of their own.

3. The document on google containing the module project is necessary for students to understand what they are required to do to complete this module. It contains the requirements of the project and also includes recommendations and tips to help students on their way.

4. The interviews with IT startups are meant to inspire students. Interviewing smaller startups show students that many such businesses start with just an innovative idea, which they could potentially think of themselves. Furthermore, this approach introduces students to performing interviews.

5. The idea and business plan mind-map are the core idea behind the project. students must work together in groups to come up with a good business idea involving IT and then map out their idea into a business plan. A mind-map is used to give students the option of creating a light version of a business plan in which the most important topics are covered, so that they will not have to create an entire, full-length business plan. By doing this, students learn to work together in order to make something of a joint idea, in which they will potentially become partners. It also forces the students to use the knowledge gained from the google document in a practical assignment.

6. The pitch of the business idea/plan in front of the class is meant as the final presentation of this module. By sufficiently pitching their idea, they can convince their teachers and others that they have picked up on the concepts of this module and that their idea is (potentially) a good one. students also gain experience in presenting and winning over other students.

8.1.4 Assessment

How do you discover the prior knowledge and the learning outcomes (students either understand the subject matter or are confused by it) in regard to this subject?

This is done through the students' execution of the module project. By pitching their idea and mind-map in front of the class, added by any other documents made by the students, teachers can adequately assess whether the concept of entrepreneurship in IT has landed with the students.

Chapter 9

Discussion

9.1 Findings

The first year of the Coderclass involves many of the concepts and skills students from all over the world learn about in secondary CS education. However, this study covers just their first year. The actual, 6-year education will contain much more than what is analyzed in this study and would thus be a better contribution to the many discussions on CS education and the problem of the many new CS curricula, by comparing learning objectives or content to other CS curricula.

Some of the findings were a bit surprising. The learning outcomes of the Coderclass show a high focus on programming concepts, which was expected, given the name "Coderclass". However, the low focus on society concepts was surprising because society has been a rather big topic in the Dutch CS community lately (particularly concerning privacy). Furthermore, it is surprising to see that there is such a low focus on Intelligence and robotics in the first year. However, this is picked up later in the course.

The findings of the learning objective analysis are also very much in line with some of the findings of the conceptual analysis. For example, it becomes clear that the core domain of programming has a huge presence in the learning goals, which is in line with the high amount of programming concepts found in the conceptual analysis. Furthermore, the learning objectives that were determined by looking into the module requirements and assignments came surprisingly close to the list of elements defined by Grover and Pea (2013) as the definition of computational thinking. One of the initial goals of this study was to determine if CT would be involved in the Coderclass at all, but the degree it did arise in was unexpected.

Student attitudes were more difficult to determine. The initial plan was

to determine attained learning outcomes rather than just finding student attitudes. By looking at the badges achieved by the students, the plan was to determine which of the requirements of each module were mastered. This turned out to be unfruitful, as the overview of badges gained by Coderclass students was unfinished by the time of analysis, and contained far more than the basic first-year modules and students at the time of completion of this study. For this reason, student attitudes through learner reports have instead been used as the main source of determining a form of (self-reported) learning outcomes.

The difficulty of using learner reports as the main source was that not every student filled them in a way that was necessary for the analysis. Many students failed to see (or report) the relevance of their learned skills in their answers, and have stated some exact things that they learned. While this has somewhat helped determine what they have learned, it has kept the Relevance attitude category very low on statements, while there was plenty to be said about Confidence, Satisfaction, and Judgement.

9.2 Limitations of the study

There are some limitations to the validity of this study. First, it must be noted that the conceptual analysis of the Coderclass has been an analysis of complete course documents for the first year of the course, while the other curricula that have been used for comparison in this thesis have been analyses of curricula documents (guidelines rather than actual learning material) of a complete curriculum. For this reason, the Coderclass has much more total concept occurrences than the other curricula. This fact makes it hard to compare the Coderclass with the other curricula accurately. However, for the purpose of this thesis, they have still been compared by comparing the percentages of the number of concepts in each knowledge category, because this gives a decent overview of what the Coderclass focuses on in their first year, especially on what their strengths and weaknesses are in specific knowledge categories, as opposed to different curricula.

Furthermore, this study has focused on the first year of the Coderclass curriculum only. An analysis of the full 6-year course will have to be performed to paint a full picture of the Coderclass. However, due to the novelty of the Coderclass, there is no material and limited plans for the later years. By the time of completion of this study, they will have just started their third year and have used their experience and feedback thus far to update the modules and plans for their first year as well. While the main learning line of the first year has remained the same, some of the analyzed modules have moved to the second year or have changed into an

optional module (such as the Raspberry PI-0 module).

This brings up the next limitation of the analysis. During this study, only the obligated modules (at the start of the first year) have been used in the analysis. There are optional modules available to the students as well. They must have chosen and completed at least one of them in their first year. These optional modules have been omitted from the analysis because students will not have used most of them during their first year.

9.3 Future research

Future curricular research on the Coderclass could be performed when their curriculum is fully formed for the full 6-year course. By doing this, a better overview of content and comparison to other CS curricula can be given. This thesis could then serve as the basis of an approach to determine the intended and attained Coderclass curricula.

9.4 Recommendations to Coderclass

The Coderclass can use the results from this study as input to shape the remaining years of their curriculum. From the conceptual analysis, it is possible to determine which of the knowledge categories has a low focus during the first year. This knowledge could be used as input for shaping the curriculum for the next couple of years. In particular, the Coderclass could create more content that focuses on society, usability and security (and possibly engineering), as these knowledge categories are the main ones in which the Coderclass scores lower than the general Dutch CS curriculum. Furthermore, mathematics and intelligence are not particularly focused on during the first year. However, other CS curricula do not focus on these categories much either. On the other side of the spectrum, the Coderclass scores exceptionally well in the categories of data, networking, modeling, and above all, programming. Thus, the Coderclass does not particularly need to focus on these categories any more than they already do.

On the subject of learning goals, the results of this study show that most of the learning goal domains of the 2016 Dutch CS curriculum are present, but that the Coderclass has a lower focus on the domains of architecture and interaction, particularly on the subdomains of ethics, automatons, security, and privacy. The Coderclass could use these results to create new learning goals involving these subdomains to process in future modules. Furthermore, learning goals that are more reflective in nature, such as learning/study reflection and orientation to profession and study are not strongly present, but this is to be expected from a first year's curriculum.

Teaching computational thinking to students is an import topic in recent CS curricula. From the results of this study, we can state that the Coderclass is well on their way of doing this in their curriculum and should continue to do so.

Finally, the attitude outcomes reported by Coderclass students tell that students are generally very positive and happy with the Coderclass in its current form. Some students were slightly more negative about the general difficulty, but this is to be expected from any curriculum. The learner reports filled in by the students did not contain many statements regarding the relevance of the thing they learn, and could possibly be taken into account for other modules. However, it is also possible that students did not understand the instructions for filling in the learner reports. Furthermore, several students seem to agree on the fact that 5 hours a week is not always enough, which could be pointing out an overfilled year plan. Finally, several students seem to agree on disliking the Raspberry PI module, as this is the only module specifically named in the negative satisfaction statements. The Coderclass could work on making this module more enjoyable.

Appendices

Appendix A

List of categories and their concepts

Here every concept and term used in the analysis of occurrences is listed as explained by section 4.1, grouped by each knowledge category. This list was taken from Steenvoorden, 2015.

Algorithms: algorithm, algorithm representation, algorithm sharing, ambiguity, breadth first search, complexity, component, computationally unsolvable, concurrency, data processing, deadlock, decision, decomposition, depth first search, finite state machine, heuristic algorithm, information sharing, input, instruction, instruction sequence, instruction set, iteration, live lock, merge sort, output, parallel processing, parallel stream, parallelisation, pattern, performance, precision, problem solving, recursion, redundancy, repetition, resource, search algorithm, selection, sequence, sort algorithm, steps, task, tractability

Architecture: apis, architecture, assembly code, binary form, binary switch, bit, byte, chip, circuit, communication layer, compiler, computer, computer component, cpu, device, digital machine, digital value, electronic device, embedded system, emulator, execute, execution model, file io, file system, flip-flop, hand-held technology, hard disk, hardware, hardware component, hardware problem, hardware sharing, instruction representation, interpreter, logic circuit, logic gate, low level language, machine, memory, mobile device, monitor, moore's law, mouse, numeric value, operating system, overflow, peripheral, personal computer, physical layer, processor, real time system, register, sampling, scheduling, single event system, software, system, system controlling, system design, thread, translation, virtual machine, von neumann architecture

Data: array, audio format, big data, binary representation, character,

character representation, compression, data, data error, data representation, data set, data storage, data type, data value, database management system, digital data, document format, file format, floating point, fraction, fraction representation, hexadecimal number, hexadecimal representation, image representation, information, information persistence, information representation, information system, integer, list (data structure), lossless compression, lossy compression, mark-up language, number representation, persistence, query language, relational database, relational schema, representation purpose, representation sharing, retrieving information, sampling frequency, signed integer, sound representation, storage, string, table, text representation, two dimensional array, unsigned integer, word

Engineering: chart, clarity, collaboration, communication, correctness, data analysis, development instrument, digital artifact, documentation, evaluation, feedback, flowchart, functional design, implementation technique, instances, object-oriented design, problem, problem exploration, problem statement, productivity tool, project, project management, prototype, refactor, requirement, scale, separation, software creation, software development (process), software life cycle process, solution, specification, stakeholder, teamwork, technology resource, technology tool, test case, tool, validation, verification, version control system

Intelligence: artificial intelligence, computer vision, human intelligence, intelligent behavior, language understanding, machine intelligence, robot component, robotics

Mathematics: and, binary number, boolean, exclusive-or, graph, logic, logical expression, not, or, quantization, set, statistical function, tree, truth table

Modelling: model, simulation

Networking: authentication, bandwidth, browser, client-server model, communication between machines, cookie, data communication, domain name service, error correction (network), fault-tolerance (network), firewall, http request, hyper link, internet, internet service, internet vs web, ip address, latency, mac address, mail header, network, network address, network component, network connection, network diagram, network functionality, network message, network path, network protection, network structure, network traffic, on-line resource, packet, packet switching, path (network), peer-to-peer, point to point transmission, protocol, queue (network), receiver, routing, search engine, search engine ranking, search query, server, server capability, shared resources (network), spooler (network), transmitter, url, web, web browser, web page, web page structure, web request, web site, web site address, web

site name

Programming: application, argument (of function), arithmetic operation, assignment, behaviour (of code), boolean operation, bug, class, conditional, conditional jump, constant, context (of application), control structure, data structure, divide by zero, efficiency, error, expression, function, high-level language, html, language, logical operation, looping (programming), method, mobile computing application (programming), paradigm, parameter, procedure, program, program creation, programming language, programming technique, readability (code), recursive function, scope, semantic error, signature, statement, string manipulation, syntactic error, usability (code), variable

Security: access rights, cryptography, encryption, password, protection, secure storage, secure transaction, security, web safety and security

Society: appropriateness, bias, career, commercial software, comprehensiveness, digital rights, ethical behaviour, experience, expression (communication), free software, hacking, information right, interdisciplinary, international network (society), law, legal behaviour, limitation of digital machines, open source development, open source software, ownership (privacy), personal information, privacy, productivity technology, proprietary software, public domain software, relevance, software license, software piracy, technology

Usability: adaptability, human computer interaction, user, user dialogue

Appendix B

Entrepreneurship Module Coderclass

Here, a representation of the entrepreneurship module is depicted as offered as a choice module in the Coderclass. It is in dutch, as the Coderclass is a dutch course. Only the overview of the module is given, as the content is the property of the Coderclass.

B.1 Module Ondernemerschap-0

Deze module is bedoeld voor groepen van 2 tot 4 personen. De lesbrieven kan je eventueel wel zelf doornemen.

In de module Ondernemerschap-0 maak je kennis met ondernemerschap in de ICT. Door middel van een aantal interviews met (startup) bedrijven ga je leren hoe je van een idee een geld verdienend bedrijf kunt maken. Je gaat leren hoe je een bij een innovatief idee een bedrijfsplan kunt maken en hoe je dit idee vervolgens kunt pitchen (een verkooppraatje geven) om mensen te overtuigen dat jouw idee een briljant idee is.

Deze module bestaat uit 2 onderdelen:

1. Lesbrieven waarin je de begrippen, tips en tricks leert.
2. Miniproject waarin je zelf een innovatief idee gaat uitwerken met behulp van een bedrijfsplan, je idee gaat pitchen en interviews gaat doet bij bestaande (startup) bedrijven.

Lesbrieven
Onderstaande lesbrieven zijn bedoeld als introductie en achtergrondmateriaal voor het miniproject. De opdrachten hierin hoef je dus niet in te leveren, maar ze zijn leuk om met je groepje te bespreken of zelf over na te denken en helpen je bij de uitvoering van het miniproject.

1. Lesbrief 1: Ondernemerschap
2. Lesbrief 2: Het bedrijfsplan
3. Lesbrief 3: Interviews

Miniproject - je eigen idee uitwerken
Miniproject: Het uitwerken van een innovatief idee tot een plan en pitch.

Links naar handige documentatie
http://www.mla.dreamstorm.nl/ - E-learning inleiding ondernemersschap
https://www.ikgastarten.nl/ondernemingsplan/ondernemingsplan-voorbeelden/de-9-bouwstenen-van-het-business-model-canvas - Een uitgebreide uitleg over het maken van een bedrijfsmodel.
https://www.bnr.nl/nieuws/ondernemen/10321413/klokhuis-en-kvk-helpen-kinderen-ondernemen - Ondernemen bij jeugd, inclusief leuke geluidsfragmenten van leeftijdsgenoten die al ondernemen!

Wanneer verdien je de badge?
Als je het miniproject succesvol hebt uitgevoerd (waarbij je interviews met bedrijven hebt gedaan, een mindmap van je bedrijfsplan hebt opgesteld en je jouw idee succesvol hebt gepitched voor de klas) verdien je deze badge.

References

Barendsen, E., Grgurina, N., & Tolboom, J. (2016). A new informatics curriculum for secondary education in the netherlands. In *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 105–117).

Barendsen, E., & Henze, I. (2015). Teacher knowledge and student attitudes in context-based science education. *NARST Annual International Conference*.

Barendsen, E., & Steenvoorden, T. (2016). Analyzing conceptual content of international informatics curricula for secondary education. In *International conference on informatics in schools: Situation, evolution, and perspectives* (pp. 14–27).

Barendsen, E., & Tolboom, J. (2016). *Advies examenprogramma informatica havo/vwo.* slo, nationaal expertisecentrum leerplanontwikkeling.

Binkley, M., Erstad, O., Herman, J., Raizen, S., Ripley, M., Miller-Ricci, M., & Rumble, M. (2012). Defining twenty-first century skills. In *Assessment and teaching of 21st century skills* (pp. 17–66). Springer.

Council, N. R., et al. (1996). *National science education standards.* National Academies Press.

De Groot, A. (1980). *Over leerervaringen en leerdoelen.[about learning experiences and teaching goals] in handboek voor de onderwijspraktijk, 10 (november), b. 1-b. 18.* Deventer, The Netherlands: Van Loghum Slaterus.

Eijk, G. H. A. V. (1984). The writing of learning experiences as a learning tool. In R. Langdon, K. Baynes, & P. Roberts (Eds.), *Design education: Proceedings of the design policy conference* (Vol. 5, p. 42-45). London.

Grover, S., & Pea, R. (2013). Computational thinking in k–12 a review of the state of the field. *Educational Researcher*, *42*(1), 38–43.

Keller, J. M. (1984). The use of the arcs model of motivation in teacher training. *Aspects of educational technology*, *17*, 140–145.

Klop, T. (2008). *Attitudes of secondary school students towards modern biotechnology.*

Schmidt, V. (2008). Vakdossier 2007 informatica. *Enschede, SLO*.

Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition.

Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher*, *15*(2), 4–14.

Shulman, L. S. (1987). Knowledge and teaching: Foundations of the new reform. *Harvard educational review*, *57*(1), 1–23.

Steenvoorden, T. (2015). *Characterizing fundamental ideas in international computer science curricula* (Unpublished master's thesis). Radboud University Nijmegen, Faculty of Science.

Tolboom, J., Grgurina, N., et al. (2008). The first decade of informatics in dutch high schools. *Informatics in Education-An International Journal*(Vol 7_1), 55–74.

Tolboom, J., Krüger, J., & Grgurina, N. (2014). *Informatica in de bovenbouw havo/vwo: Naar aantrekkelijk en actueel onderwijs in informatica.* slo, nationaal expertisecentrum leerplanontwikkeling.

Van den Akker, J. (2004). Curriculum perspectives: An introduction. In *Curriculum landscapes and trends* (pp. 1–10). Springer.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.