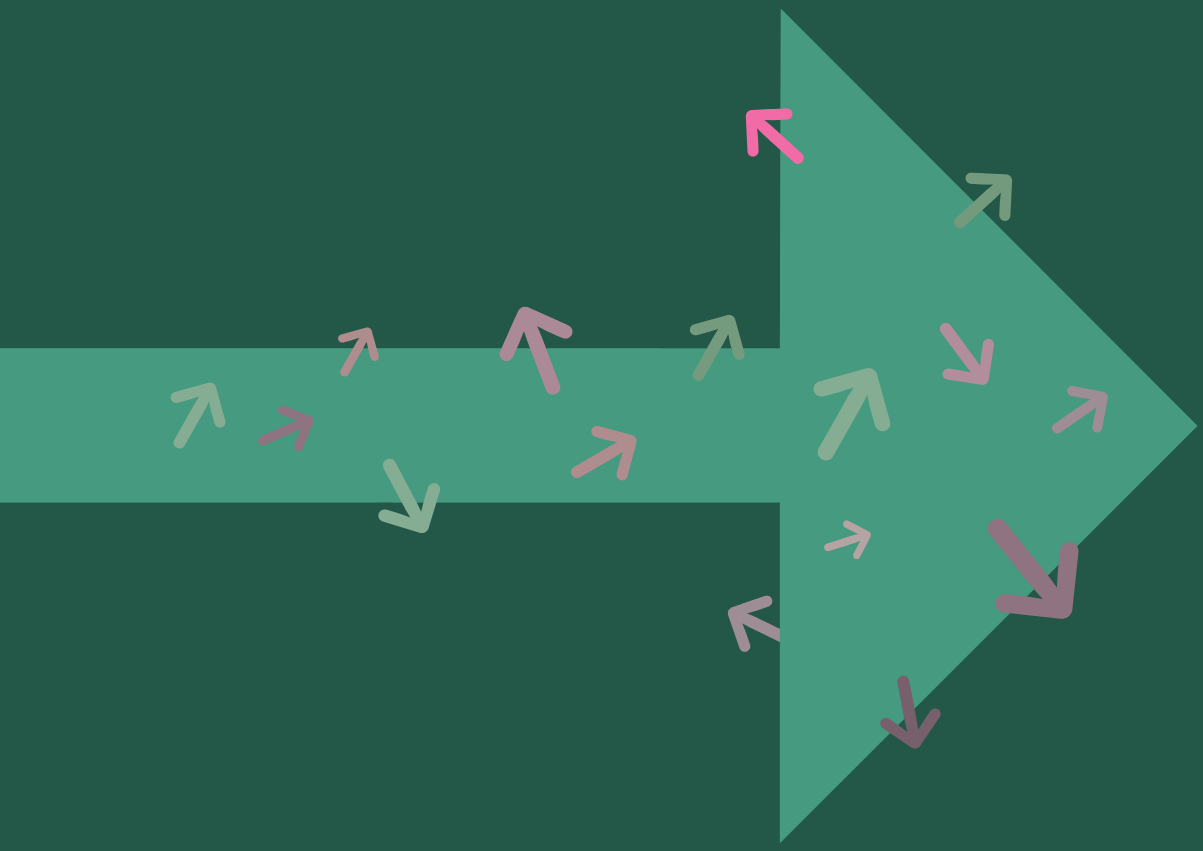
Getting the Picture



Modeling and Simulation in Secondary Computer Science Education

Nataša Grgurina

Getting the Picture

Modeling and Simulation in Secondary Computer Science Education

Nataša Grgurina

Layout: Ferdinand van Nispen, *my*-thesis.nl
Cover design: Rein Scholte, www.rwscholte.com
Print: GVO drukkers en vormgevers, proefschriften.nl, Ede, NL

Digital version: https://www.my-thesis.nl/grgurina/

The work on this research project was supported by the The Netherlands Organisation for Scientific Research grant nr. 023.002.138.

university of groningen

Getting the Picture

Modeling and Simulation in Secondary Computer Science Education

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. C. Wijmenga
and in accordance with
the decision by the College of Deans.

This thesis will be defended in public on

Thursday 28 October 2021 at 16.15 hours

by

Nataša Grgurina
born on Februari 23, 1966
in Zagreb, Croatia

Supervisors
Prof. K. van Veen
Prof. E. Barendsen
Prof. B. Zwaneveld

Assessment Committee
Dr. M. Helms-Lorenz
Prof. C. Schulte
Prof. V. Dagienė

Table of contents

List of figures

List of tables

Preface

As a beginning computer science (CS) teacher in secondary school, I had a vague notion that familiarity with computer science was going to be important for my students because it was going to somehow empower them, and fun to do anyway. During the two decades since, computer science, its impact on all aspects of our lives, and, ultimately, my thinking about the goals and aims of teaching computer sciences evolved drastically.

Throughout my career as a computer science teacher in secondary education, my primary drive has always been to provide my students with knowledge, skills and a curious mindset which would be useful to them for the rest of their lives. I find teaching modeling and simulation to be perfectly aligned with this goal.

In this thesis, I describe my journey to explore pedagogical aspects of teaching modeling and simulation within the elective Computer Science course in the upper grades of secondary education in the Netherlands.

Chapter 1

Introduction

This chapter introduces the research aim of this thesis and outlines its content. First, we give a brief overview of computer science education and look at computational thinking (CT) and its relation to computer science (CS). We introduce four components of content specific pedagogy as a lens through which we look at teaching CT and then motivate and explicate our research questions. Finally, we present the overview of the thesis.

1.1 Computer Science Education

Since the emergence of computers in the 1950's, learning about them is considered important: hence the introduction of computer science (CS) education in K-12 worldwide. In this introductory chapter, we portray the aims of teaching CS, illustrate these with examples of CS education in K-12 from several countries, and describe the exploding interest in teaching some aspect of CS to all students with the reintroduction of the notion of Computational Thinking (CT) in the 2000's. We then zoom in on modeling and simulation, an aspect of CT that has barely been touched upon in the context of Computer Science Education Research (CSER) and describe how we look at teaching modeling and simulation in secondary CS education in the Netherlands through the lens of pedagogical aspects of teaching a particular subject that is derived from the notion of Pedagogical Content Knowledge (PCK).

So, why do we teach computer science (CS)? The purpose to teach anything can be seen as threefold: (1) *qualification* — to provide students with knowledge and skills to enable them to do something, (2) *socialization* — to become part of existing culture and tradition, and (3) *subjectification* — to develop autonomous and individual thinking and acting (Biesta, 2015).

The rationales for teaching CS evolve together with CS itself. In the early days of computing in the 1950's, when computers were scarce and difficult to use, the focus lay on training for technical jobs — thus providing qualification to students. With the increased development and availability of computers in the second half of the 20th century, the focus shifted to training for software development and use in academia. Nowadays, when computers in all possible shapes and forms permeate every pore of our professional, social and private life, the socialization and subjectification purpose of learning CS are gaining significance. The motives to teach CS refer not only to preparing students for the labor market, but also to promoting computational thinking (i.e. "computer scientists' ways of thinking, heuristics and problem-solving strategies") and computational literacy (i.e. "a set of material, cognitive, and social elements that generate new ways of thinking and learning"), supporting equity of participation (Blikstein & Moghadam, 2019; Vogel et al., 2017), as well as bringing up broader issues of citizenship and civic life; scientific, technological and social innovations, school improvement and reform, and finally, fun, fulfillment and personal agency (Vogel et al., 2017).

To meet these needs, computer science is taught worldwide in K-12 education in many various forms in an increasing number of countries. In some cases, as an independent subject, either as a compulsory subject or as an elective, and in other cases integrated in other school subjects. Furthermore, as various as the motives are to teach it, so are the interpretations of what is understood under CS. In our view, CS — often referred to as *informatics* or *computing science* as well— is the discipline dealing with scientific and mathematical approach to information processing and computation, and design of computing machines. However, when it comes to teaching CS, as we will see from several examples below, the interpretations of what is understood to be CS vary greatly. CS is considered to be about the scientific discipline — in line with our definition, or about digital and computer literacy; or about computational thinking, information and communication technologies; or about a combination of these (Guerra et al., 2012).

Illustrative are the examples of countries with various forms of CS education.

Many East European countries introduced CS into secondary schools in the 1980's. The aims of teaching CS and the curriculum evolved greatly since then. For example, in the current Lithuanian curriculum, CS is a compulsory subject in grades 5 - 10 and its focus lies on digital and computer literacy. Additionally, there are elective courses on algorithms and programming in the higher grades of secondary school (Dagienė & Stupuriene, 2016b).

In Croatia, a lot has changed since the 1980's with first programing lessons in BASIC and Pascal on the four computers available in a progressive school specializing in math and computer science. In 2016, an expert group proposed a new comprehensive CS curriculum spanning all grades of primary and secondary education and covering four domains: information and digital technologies, computational thinking and programming, digital literacy and communication, and finally, e-society (Brodjanac et al., 2016).

In Denmark, CS was a secondary school subject since the late 1960's. As in other countries, it evolved greatly since then, and in the current secondary school curriculum, first implemented in 2011, its content is described through seven knowledge areas: importance and impact, application architecture, digitization, programming and programmability, abstraction and modeling, interaction design, and finally, innovation (Caspersen & Nowack, 2013a). In 2016, CS entered all Danish secondary schools, as a compulsory subject in certain types of secondary schools and as elective in others. A year later, primary schools followed with CS

being an optional subject in grades 7-9 (Caspersen & Nowack, 2013a; Vahrenhold et al., 2017).

In Germany, with its sixteen federal states, each with their own educational system, the situation concerning teaching CS varies per state. In Bavaria, for example, CS has been taught since the 1960's and currently, as of 2004, it is taught in secondary education in grades 6 12 in grammar schools (i.e. gymnasia). In the 6th and 7th grade it is compulsory for all students, and in the 9th and 10th grades for students in the science and technology track. Additionally, there are elective CS courses in the 11th and 12th grade. These courses focus predominantly on computing as scientific discipline (Hubwieser, 2012).

After the dramatic appeal in the Royal Society of England report (Furber, 2012), the school subject called *Computing* was introduced in England in 2013 as a compulsory subject for all students in primary and secondary schools. This subject is about computer science, information technology and digital literacy (Barendsen et al., 2015).

France is in the process of introducing CS into primary schools, as an integral part of math and technology courses. In 2012, a computer science course was introduced for scientific Baccalauréat students only, and since 2014 a broader computer science course was offered as an optional subject in other types of secondary education. Yet, the French Academy of Sciences in their 2013 report expresses the concerns that "In the computing field, Europe and France in particular are far behind, both conceptually and industrially, compared to more dynamic countries such as the United States and certain Asian nations" and recommend compulsory CS courses emphasizing the concepts, science and techniques of computing to be introduced into primary, secondary and tertiary education (*Teaching computer science in France: Tomorrow can't wait*, 2013). The newest development is that in 2020, the two courses in France are replaced by a new compulsory course in grade 10 — Digital Sciences and Technology (in French: Sciences numériques et technologie (SNT)) and an elective specialist course in grades 11 and 12 — Digital and Computer Sciences (in French: Numérique et sciences informatiques (NSI)) (*School Education in France - Éduscol*, 2020).

In the USA, local authorities are in charge of education, and teaching CS varies greatly from state to state and within the states. In 2010, a joint report written by Computer Science Teacher Association (CSTA) and Association for Computing Machinery (ACM) expressed great concern about the failure to teach CS and recommends that CS becomes a core academic subject with a curriculum focusing

on algorithmic/computational thinking concepts (Wilson, 2010). Subsequently, the federal government has put forward and funded a number of initiatives to support and advance the CS education, for example *Every Student Succeeds Act* (Every Student Succeeds Act (ESSA), 2015) and President Obama's *CS For All* (*CS For All*, 2016).

In the Netherlands, CS has been an established elective subject in the higher grades of secondary education since 1998, as described in detail in chapter 2. In addition to this subject focusing on computing as a scientific discipline, in the lower grades of secondary education, in the late 1990's and 2000's, there used to be a course focusing on ICT which often got integrated in other courses and in the long run died out. In 2012, The Royal Netherlands Academy of Arts and Sciences (in Dutch: De Koninklijke Nederlandse Akademie van Wetenschappen, further abbreviated as KNAW) published a report expressing their concerns about teaching CS and recommended to not only overhaul the existing elective CS subject, but also to introduce a new compulsory subject *Information & communication* in the lower grades of secondary education (KNAW, 2012). The Dutch situation is described in greater detail in chapter 2.

The situation in the few countries described here is illustrative of the evolution of CS education both from the point of view of institutionalized education reforms — ranging from no CS education to introduction of mandatory CS education for all students, as well as the related intertwined changing and evolving aims and objectives of teaching CS — from specialist professional training to fundamental life skills. We also see that, as omnipresent and multifaceted the computers and their usage are in our modern world, so is the discussion about the necessity and aims of teaching CS and the position CS courses get in the curriculum. This great variation notwithstanding, the spirit of time is clearly visible in the desire to make the CS education available to all students in K-12 and to focus on computational problem-solving (Tedre et al., 2018).

This spirit, expressing the desire to empower all students by teaching them certain aspects typical for CS, was captured in the 2006 seminal article by Wing who rekindled the notion of computational thinking (CT) first introduced by Papert (1980) by asserting that, "to reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" (Wing, 2006). Wing globally sketches what CT is and what it is not in her view and stresses that it is about attitude and a skill set for everyone, not just computer scientists.

CT brings together the subject matter from a particular scientific discipline — or even from everyday life — seeking to solve a particular problem or find an answer to a question, and computing which helps solve that problem or find the answer. This CT problem-solving process involves three steps (Barendsen & Bruggink, 2019). First, that problem or question is expressed in computational terms such as data or processes, thus allowing for use of computing to solve it. Second, a computational solution is constructed, either by using existing applications or by devising new algorithms and writing new programs. Essential to the nature of CT is that this solution should be executable (Martin, 2012). Finally, that computational solution is interpreted in terms of the original subject matter, thus providing the solution to the original problem or answering the question.

Wing's perceived need to teach CT struck a chord with educators and researchers who sought to formulate a precise description of this concept and devise ways to teach it.

1.1.1 Definitions of Computational Thinking

There have been numerous efforts to obtain a clear-cut definition of computational thinking.

In 2010 in the USA, the National Research Council held a workshop on the nature and scope of Computational Thinking (CT). While there was a broad consensus on the importance of (teaching) CT, the workshop did not result in an exclusive definition of this concept (Thinking & Council, 2010). The Computational Thinking Task Force of the Computer Science Teachers Association (CSTA) in the USA did, however, suggest an operational definition of CT tailored to the needs of K-12 education. In their framework, they describe CT as follows:

CT is a problem-solving process that includes (but is not limited to) the following characteristics:

- *Formulating problems in a way that enables us to use a computer and other tools to help solve them*
- *Logically organizing and analyzing data*
- *Representing data through abstractions, such as models and simulations*
- *Automating solutions through algorithmic thinking (a series of ordered steps)*
- *Identifying, analyzing, and implementing possible solutions with the goal of achieving the most efficient and effective combination of steps and resources*

- *Generalizing and transferring this problem-solving process to a wide variety of problems.*

In addition to this definition, they touch upon necessities for students' learning of CT and add:

These skills are supported and enhanced by a number of dispositions or attitudes that are essential dimensions of CT. These dispositions or attitudes include:

- *Confidence in dealing with complexity*
- *Persistence in working with difficult problems*
- *Tolerance for ambiguity*
- *The ability to deal with open-ended problems*
- *The ability to communicate and work with others to achieve a common goal or solution.*

Furthermore, a vocabulary of CT is supplied, describing CT in terms of its core concepts: *data collection, data analysis, data representation, problem decomposition, abstraction, algorithms & procedures, automation, simulation and parallelization* (CSTA Computational Thinking Task Force, 2011).

This view of CT as a problem-solving process puts emphasis on the construction of a computational solution for a given problem after that problem is expressed in computational terms.

While this definition is intended to portray CT across disciplines and is by no means meant to be limited to CS, there are also initiatives to define CT specifically with CS in mind.

The Carnegie Mellon Center for Computational Thinking (CMCCT), with its mission to develop computing research, developed a CT framework describing CT as consisting of three aspects: *making use of abstraction and modeling, thinking algorithmically,* and *understanding scale* (*Carnegie Mellon Center for Computational Thinking*, 2010).

Brennan and Resnick observed activities of Scratch programmers and derived a definition of CT through its three dimensions: *computational concepts* (i.e., sequences, loops, events, parallelism, conditionals, operators and data), *computational practices* (i.e. being incremental and iterative, testing and debugging, reusing and remixing, and, abstracting and modularizing), and *computational perspectives* (i.e. expressing, connecting and questioning) (Brennan & Resnick, 2012).

In an effort to find common ground in various definitions of CT, Selby and Woollard (2013) describe CT as "a focused approach to problem solving,

incorporating thought processes that utilize abstraction, decomposition, algorithmic design, evaluation, and generalizations" — thus emphasizing those steps of CT problem-solving process which express the original problem in computational terms and interpret the computational solution in the domain where the problem originates (Barendsen & Bruggink, 2019).

The discussion about the precise definition of CT is still going on (Grover & Pea, 2018; Guzdial, 2018) and there are many authors who express their vision about the importance of CT and the definition as they see it (Allan et al., 2010; Caspersen & Nowack, 2013b; Fletcher & Lu, 2009; Henderson, 2009; Hu, 2011; Kafai & Burke, 2013; Malyn-Smith et al., 2018; Wing, 2006, 2008, 2014).

However, not everyone is convinced about the idea of CT. Some authors are troubled by the lack of consensus on a precise definition of CT and the unresolved question on how exactly is CT different from, for example, mathematical thinking (Jones, 2011) or problem solving (Glass, 2006). Hemmendinger (2010) warns not to get carried away with the newest fad and says many elements of CT are not unique or exclusively reserved for CS. Denning (2009) agrees and sees another problem with CT: that it might be seen as characterization of CS, which is most definitely not the case in his view; "Computational thinking is one of the key practices of computer science. But it is not unique to computing and is not adequate to portray the whole of the field." Then, together with Tedre (2016), he goes on to examine a number of threats to CT initiatives, as so does Guzdial (2015) from a practical point of view.

1.1.2 Instructional Approach and Assessment

Regardless of the critical sounds, CT has gained a huge momentum and there are many initiatives to weave it into the school curricula. The idea that CS education — which could arguably be considered a natural habitat for CT — and more specifically, various forms of programming education — could contribute to the development of students' CT is very common (Bers et al., 2014; Davies, 2008; Gouws et al., 2013a; Grover, 2011; Howland et al., 2009; Kafai et al., 2013; C. C. Selby, 2014; Walden et al., 2013; Weintrop & Wilensky, 2013). However, Lu & Fletcher (2009) add that, conversely, students proficient in CT might be more inclined to major in CS and there are even initiatives to teach a CT for CS course to students without prior CS knowledge (Kafura & Tatar, 2011).

Looking from a different perspective, since CT can form a bridge between CS and an application domain, there are numerous suggestions to employ CT

to advance the learning of science outside the CS education. Examples include incorporating CT into middle school life science classes (Cateté et al., 2018; Gendreau Chakarov et al., 2019) and then using sensors to observe mold growth (Gendreau Chakarov et al., 2019); or, for science majors, by integrating CT into a bioinformatics course (Qin, 2009) or by focusing of computational principles in scientific inquiry (Hambrusch et al., 2009). Learning science can be supported through the use of modeling and simulation — an integral aspect of CT — too: for example, by developing computational models and simulation for science in grades 4-6 (Basu et al., 2013, 2014; Dwyer et al., 2013) or for physics in grade 9 (Aiken et al., 2012), often by employing tailor-made software (Basawapatna et al., 2013). It is suggested that CT can help put modeling — a core aspect of engagement in science (Justi & Gilbert, 2002) — within the reach of K-12 students (Sengupta et al., 2013; Wilensky, 2014; Wilensky et al., 2014). Teaching CT found its way also into, for example, games where children specify the algorithms describing the behavior of the characters in a game (Weller et al., 2008); music with musical live coding in Scratch (Ruthmann et al., 2010), and journalism where middle school students together with their teachers develop news stories and present them as text, video and animations in Scratch (Wolz et al., 2010). There are also suggestion to promote CT through contests. Bebras is an international contest for primary and secondary schools where tasks are categorized according to concepts they cover and it is suggested they can be incorporated into curriculum to promote CT (Dagienė & Sentance, 2016).

In parallel with these specific endeavors, comprehensive frameworks are being developed to inform and guide the integration of CT into K-12 curricula, with special attention given to classroom techniques, focusing on instructional approach. For example, by introducing into K-6 education various CT programs, courses or modules based on generic CT framework containing CT skills abstraction, generalization, decomposition, algorithmic thinking and debugging, through a holistic design approach (Angeli et al., 2016). Curzon et al. (2014) provide a framework with examples to help teachers teach CT, consisting of four stages: (1) definition, (2) concepts (algorithmic thinking, evaluation, decomposition, abstraction, generalization), (3) classroom techniques with examples of learners' behavior, and (4) assessment which can be performed with an adapted version of assessment used for the subject Computing. For higher education, Perkovic et al. (2010) developed a framework to be used at their university *"by faculty without formal training in information technology in order to understand and integrate computational thinking into their own general education courses"* and provide

examples for CT implementation in various courses. While this example is not from K-12 education, it is illustrative of the challenges facing educators who are about to engage in teaching CT.

Lee et al. (2011) formulated suggestions for instructional strategies supporting the development of CT, too. First, they advise to have students work in rich computational environments, and second, to scaffold interactions into a three-stage progression that describe the stages of engagement of learners in these rich computational environments: *use — modify — create*. Touretzky et al. (2013) suggest a progression from simple to more complex programming language in a CS course in order to bring students closer to true CT.

Teaching goes hand in hand with assessment, so when CT finds its way into the education, the question arises how to assess it. There are already numerous attempts to do so: for example, Koh et al. (2014) developed Computational Thinking Pattern Analysis to analyze CT patterns in games submitted by students and found that a promising approach. Werner et al. (2012) looked at the games produced by their students too and based their assessment on the CT framework developed at CMCCT (*Carnegie Mellon Center for Computational Thinking*, 2010). They asked middle school students to modify and fix existing Alice programs in order to assess the first two aspect of the CMCCT framework (making use of abstraction and modeling, and thinking algorithmically) and concluded that this was a promising strategy for assessment (Werner et al., 2012). Similarly, Brennan and Resnick (2012) — after having developed their framework that describes CT in terms of *computational concepts, computational practices* and *computational perspectives* — tried three approaches to assess the development of CT of young people programming in Scratch. They describe strengths and limitations of these approaches, conclude that none of them was particularly effective, and finally formulate six suggestions for assessing CT via programming (Brennan & Resnick, 2012):

1. assessment should support further learning
2. creating and examining projects should be an integral part of the assessment
3. the project designer should illuminate the design process
4. (formative) assessment should take place at multiple moments during the project development
5. value multiple ways of knowing
6. include multiple viewpoints.

Lye and Koh (2014) then used the framework suggested by Brennan and Resnick — describing CT in terms of *computational concepts, computational practices* and *computational perspectives* — to analyze 27 intervention studies about development of CT in CS courses in K-12. Their findings were that most of these interventions focus on computational concepts while computational practices and perspective barely get any attention, and they go on to recommend an instructional approach described as *"a constructionism-based problem-solving learning environment, with information processing, scaffolding and reflection activities, could be designed to foster computational practices and computational perspectives."*

1.1.3 Teachers

With all this attention to CT, it is important to facilitate the teaching itself. There are voices expressing concerns whether teachers are ready to teach CT without specific preparation (Perković et al., 2010). Bort and Brylow (2013) set out to measure the integration of CT core concepts into lesson plans of the teachers attending their Computer Science for High School (CS4HS) workshops and found that, while the teachers were enthusiast, there was ample room for improvement of the lesson plans the teachers produced. Digging deeper, Czerkawski (2013) surveyed six instructional designers on their ideas how to promote the ideas of CT in the curriculum, with instruction based on ADDIE model (Analysis, Design, Development, Implementation and Evaluation) and particular emphasis on analysis and design phases. The findings are described in terms of: dispositions and characteristics of the learners, teaching strategies, learning outcomes, pedagogical considerations, adult learning considerations, user experience, instructional prototype & curriculum design, and finally, visual and multimedia design. Together with Xu, they provide a sample activity plan for CT in educational technology courses (Czerkawski & Xu, 2012). Yadav et al. (2014, 2011) observed that most of the efforts to familiarize teachers with CT are focused on CS teachers and turned their attention to pre-service teacher training of primary and secondary teachers of other disciplines. They introduce a compulsory CT module into the teacher education and observe that it results in a positive attitude of the students towards CS and integration of computing into their teaching. To further advance teacher preparation, they recommend to redesign courses on educational technology and methods to better develop future teachers' competencies in CT, and to have education and CS faculty jointly work on these efforts (Yadav et al., 2017).

As a part of growing body of research resulting in recommendations concerning teaching CT, there is also a budding interest to look specifically at CS teachers (Mara Saeli et al., 2012; Yadav & Berges, 2019) by examining their Pedagogical Content Knowledge (PCK) — subject matter knowledge *for teaching* (Shulman, 1986). PCK represents teacher's thoughts about teaching a particular topic and it can be described with various granularity, taking into account a range of circumstances influencing teaching (Carlson & Daehler, 2019; Grossman et al., 2005; Loughran et al., 2004; Magnusson et al., 1999).

The most cited model of PCK is that of Magnusson et al. (1999), defining five knowledge components of the construct of PCK. Four of these correspond to the following elements of content-specific pedagogy, which we will refer to as M1, M2, M3 and M4:

- M1: goals and objectives for teaching this particular content;
- M2: students' understanding of this content, including requirements for learning and their difficulties;
- M3: instructional strategies connected to this content;
- M4: methods of assessing students' understanding of this content.

Most definitions and operationalizations of PCK share their recognition of the above components. Magnusson et al. (1999) also proposed 'orientations to teaching science' as a fifth knowledge component. We do not include this component in our analyses, as it is considered less content-specific, and moreover, it is presented as an underlying type of knowledge influencing M1 to M4 (cf. Henze & Barendsen, 2019).

In this thesis, we will use M1 to M4 to indicate the pedagogical aspects of specific content, as well as to characterize components of teacher's PCK for teaching that content.

1.1.4 Computer Science Education Research

The above scientific developments are illustrative of the emerging Computer Science Education Research (CSER) — a research field inspired and building upon rich traditions of related research fields (Guzdial & Boulay, 2019; Malmi et al., 2010). Following the call from the students, parents, educators, institutions, governments and other stakeholders to advance CS education (Furber, 2012; Gander et al., 2013; KNAW, 2012; *Teaching computer science in France: Tomorrow can't wait*, 2013; Wilson, 2010), researchers see a role for themselves to support CS education by supplying solid theoretical underpinnings for its implementation.

To the CSER researchers, students' learning is traditionally of interest, and this topic is accompanied by interest in teaching methods, pedagogy and learnability; assessment issues and learning analytics; tools, and curricular aspects. Moving outside the classroom, we see interest in professional development of teachers; participation and equity issues, blended and informal learning experiences; and social and global challenges in CS education. Furthermore, we observe a growing demand for academic rigor in CSER and a continuous encouragement to conduct a sound academic debate on new or unresolved issues.

1.2 Situation in the Netherlands

We turn our attention to the Netherlands to explain the situation of the educational context where this project was carried out and to motivate the specific research questions.

We see that teaching CT problem-solving skills did not get enough attention from policy makers and was hardly represented in school curricula in general (Barendsen & Zwaneveld, 2010), and that the situation of the elective Computer Science (CS) course in the upper grades of secondary education was far from thriving, as described in chapter 2. However, recent developments are promising: since the fall of 2019, the Computer Science (CS) course in the upper grades of secondary education is taught according to a new curriculum (described in more detail in section 2.3.3). For the K-9 education (i.e., elementary schools and lower grades of secondary education — see figure 2), there is a plethora of initiatives to advance digital literacy, media wisdom, ICT skills, information skills, computational thinking, programming, coding, and any combination of these. One of them is the nationwide curriculum.nu[1] initiative where teachers and school administrators cooperate on a project to overhaul the whole of the K-9 curriculum and define it in terms of nine connected and coherent domains; one of them is to be the new domain *Digital literacy*.

Digital literacy is described in terms of *big assignments* that express the essence of the discipline: *Communicating and collaborating, Digital citizenship, Data and information, Using and managing, Applying and designing, Digital economy, Security and privacy,* and finally, *Sustainability and innovation*. All of these big assignments encompass four perspectives: *dealing with, thinking over, creating with,* and, *knowledge of*. These perspectives are all connected to the interpretation

1 Https://curriculum.nu

of digital literacy by the Dutch National institute for curriculum development (Dutch: Stichting leerplanontwikkeling, SLO), which is described as containing four elements: *ICT skills, Media wisdom, Computational thinking* and *Information skills* (*Computational thinking*, 2020; Thijs et al., 2014a). The big assignments are situated in three contexts: *personal life, society,* and*, education and profession.*

It is expected that this curriculum.nu initiative — and in particular its domain *digital literacy* — will fill the gap signaled in the report by The Royal Netherlands Academy of Arts and Sciences (2012) by providing the desired subject *Information & communication* in the lower grades of secondary education, albeit under a different name. It is also presumed that *digital literacy* will seamlessly tie into the Computer Science (CS) course in the upper grades of secondary education (described in detail in chapter 2), as recommended in the new 2019 curriculum for the Computer Science (CS) document (Barendsen & Tolboom, 2016).

One might think that students who follow a computer science course would become proficient at computational thinking spontaneously — a wish harbored by many a secondary school computer science teacher. Furthermore, secondary CS education should cater not only to those of the students who plan to pursue careers in computer science or some related field, but first and foremost to the majority of the students who will chose to do something else (Guzdial, 2019). Therefore, the question arises, what aspect of computational thinking tends to be underexposed in the typical CS classroom, as well as important and meaningful to all of the secondary CS students, thus deserving more of our attention. The answer is modeling and simulation — a set of new learning objectives introduced in the 2019 secondary CS curriculum in the Netherlands. As described in section 3.1, this curriculum unites modeling and simulation under the name *Computational Science*[2] and provides the following description: *"Modeling: The candidate is able to model aspects of a different scientific discipline in computational terms"* and *"Simulation: The candidate is able to construct models and simulations, and use these for the research of phenomena in that other science field."* Additionally, modeling itself is to be a part of the compulsory core curriculum, described as *"Modeling: The candidate is able to use context to analyze a relevant problem, limit this to a manageable problem, translate this into a model, generate and interpret model results, and test and assess the model. The candidate is able to use consistent reasoning." (Barendsen & Tolboom, 2016).*

2 In this thesis, we use terms *modeling*, *modeling & simulation* and *Computational Science* interchangeably, unless explicitly stated otherwise.

1.3 Research Questions

As seen in the examples in section 1.1, there are numerous attempts to employ CT aspects *modeling & simulation* within courses other than CS to support the learning of the subject matter in those courses. *Modeling & simulation* is considered to be a fundamental part of CT (CSTA Computational Thinking Task Force, 2011): modeling builds a bridge where a problem in a particular discipline meets CS by expressing the original problem in computational terms and interpreting the computational solution in the domain where the problem originates. (Barendsen & Bruggink, 2019). To run a simulation with that model, a computational solution is constructed, either by using existing applications or by devising new algorithms and writing new programs, thus clearly engaging in typical CS activities.

We embrace this idea of using CT aspects *modeling & simulation to* support the learning of the subject matter of various disciplines, but we choose to do so from within a CS course, taking the learning objectives of Computational Science in the Dutch 2019 secondary CS curriculum as our starting point.

There are ample courses that teach modeling, and ample courses that teach scientific inquiry, and ample courses that teach programming. However, we are not aware of research into teaching a combination of these in a CS course, where all three steps of the CT problem-solving process get sufficient attention — in other words, where expressing a problem in computational terms and interpreting computational solution in terms of the original subject matter get as much attention as the construction of an executable computational solution.

Considering our interest in teaching modeling and simulation, we see that a research project that looks into curricular issues, students' learning, teaching methods, assessment, and professional development of teachers regarding modeling and simulation within a secondary CS course would fit seamlessly into the contemporary CSER program. We are, then, interested in teaching modeling and simulation as described in the new Dutch curriculum, i.e., as generic scientific competences within a CS course meant to equip the students with skills to perform scientific inquiry in any discipline of their interest. We therefore engage in a research project with the main objective to *explore the pedagogical aspects of Computational Science and the CS teachers' PCK for teaching Computational Science*.

As announced earlier, we will use Magnusson's (1999) components M1 to M4 to characterize these pedagogical aspects, as well as the respective categories within teachers' PCK.

- M1 goals and objectives
- M2 students' understanding (including requirements for learning and their difficulties)
- M3 instructional strategies
- M4 methods of assessment.

We translate our main objective into four research questions:

- RQ1 What computational thinking activities constitute the problem-solving process associated with Computational Science? This question is aims to find an operational definition of the learning goals and objectives of Computational Science. (M1)
- RQ2 How can the students' understanding of modeling activities be portrayed in terms of their requirements for learning and difficulties they encounter? (M2)
- RQ3 What are characteristics of a valid and reliable assessment instrument for Computational Science? (M4)
- RQ4 How can the teachers' PCK for teaching Computational Science be portrayed in terms of the four components M1 to M4 of PCK?

We will address instructional strategies for Computational Science (M3) by designing a learning activity for Computational Science in the context of RQ3.

1.4 Structure of the Dissertation

In this section, we describe the structure of this thesis. We list the studies we performed to depict the context where this research project took place and the studies of the research project themselves, together with specific research questions they aim to answer and how the findings from particular studies inform the consequent studies.

After the introductory chapter, chapter 2 of this thesis reports on a context study portraying the birth and the first decade of the elective computer science course in higher grades of senior secondary education (in Dutch: HAVO) and pre-university education (in Dutch: VWO) in the Netherlands. It sketches the Dutch educational system, the position of computer science within the secondary

school curriculum, the objectives of this course and the intended assessment. Furthermore, the in-service teacher training for the first ever CS teachers is described together with their experiences and practices in teaching this new subject. The first part of the chapter concludes with the discussion of the many challenges and concerns faced by computer science in secondary education in the Netherlands during its first decade. The second part of this chapter describes the second decade of the computer science course in secondary schools in the Netherlands. It describes the events and processes that led to the renewal of the curriculum for this course, the curriculum itself with the principles it is based on and its aims, the current process of the teaching material development, the related research, the teacher training, curriculum reform in primary and lower secondary education, and the current situation of computer science as an upper secondary school subject, together with the challenges it still faces. This chapter is based on the articles The First Decade of Informatics in Dutch High Schools (Grgurina & Tolboom, 2008) and The Second Decade of Informatics in Dutch Secondary Education (Grgurina, Tolboom, et al., 2018).

Chapter 3 zooms in on modeling and simulation. Under the name Computational Science, modeling and simulation is included as an elective theme in the new 2019 Dutch secondary school computer science curriculum. This chapter is primarily devoted to answering our first research question related to Magnusson's component M1: What computational thinking activities constitute the problem-solving process associated with Computational Science? It presents our first study that focuses on establishing an operational description of the intended learning outcomes of Computational Science describing the activities a student engages in when exploring a phenomenon of their choice through modeling and simulation. Furthermore, it reports what data sources are found to be suitable to monitor students' learning outcomes when engaging in modeling activities — Magnusson's component M4 — thus setting the stage to answer our third research question: What are characteristics of a valid and reliable assessment instrument for Computational Science? Finally, this chapter explores what specific challenges do the students experience when engaging in modeling activities — Magnusson's component M2 — thus touching upon our second research question: How can the students' understanding of modeling activities be portrayed in terms of their requirements for learning and difficulties they encounter? This chapter is based on the paper Defining and Observing Modeling and Simulation in Informatics (Grgurina et al., 2016)

Informed by the findings from the previous study, chapter 4 describes our second study that portrays computer science teachers' initial pedagogical content knowledge (PCK) on modeling and simulation — Magnusson's component M3 — thus answering our fourth research question: How can the teachers' PCK of teaching Computational Science be portrayed in terms of the four components of PCK? This chapter is based on the paper Investigating Informatics Teachers' Initial Pedagogical Content Knowledge on Modeling and Simulation (Grgurina et al., 2017).

In chapter 5 we describe our third study focusing on the development of an assessment instrument as a part of a lesson unit on Computational Science, to monitor the levels of understanding in the learning outcomes of students engaging in modeling projects. Here we focus on Magnusson's component M4 and on answering our third research question: What are characteristics of a valid and reliable assessment instrument for Computational Science? The development of this assessment instrument and the teaching materials for the lesson unit is based on the findings of the two previous studies which provided us with an operational description of the intended learning outcomes of Computational Science — Magnusson's component M1 — and with CS teachers' initial PCK on modeling and simulation that specifically contributed to our understanding of suitable instructional strategies — Magnusson's component M3. This chapter is based on the paper Assessment of Modeling and Simulation in Secondary Computing Science Education (Grgurina, Barendsen, Suhre, Zwaneveld, et al., 2018).

Chapter 6, our final study, looks into students' understanding — Magnusson's component M2 — while they work on computation science assignments with the teaching materials which we developed using the findings of the first two studies which informed us about suitable instructional strategies — Magnusson's component M3. Here we focus on our second research question: How can the students' understanding of modeling activities be portrayed in terms of their requirements for learning and difficulties they encounter? In particular, we focus on the students' understanding of the model validation in terms of validation techniques they employ to ensure the development of valid models.

Figure 1 depicts the structure of this research project. The first study (chapter 3) informed the second study (chapter 4). Together, they informed the third and the fourth study — chapter 5 and chapter 6.

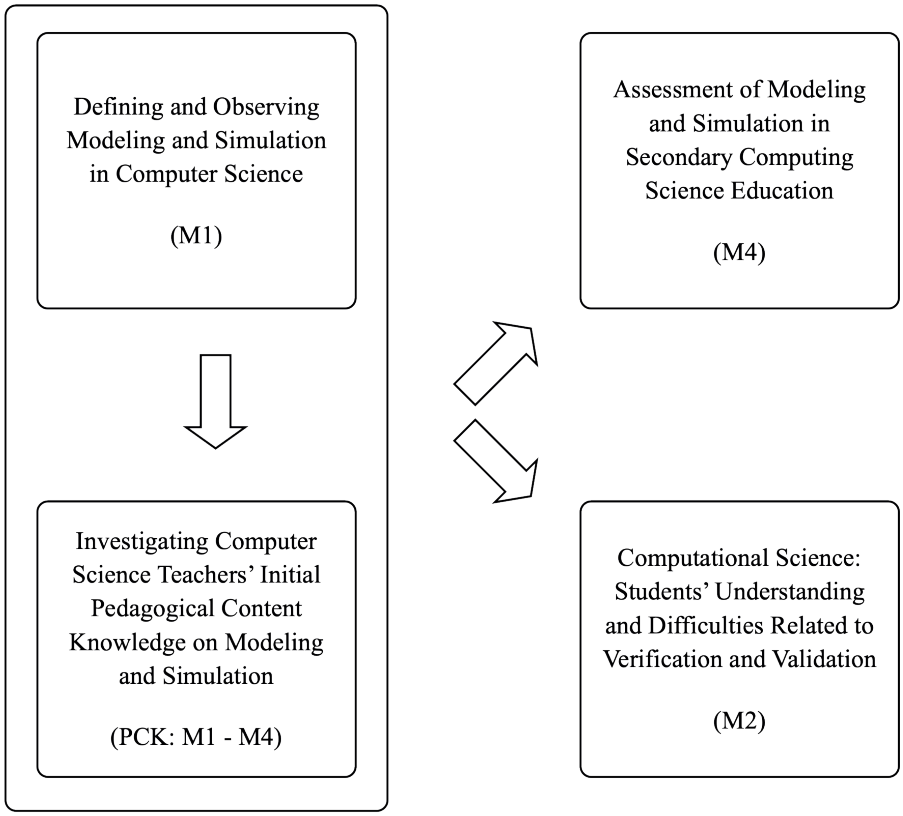


Figure 1: Structure of this research project.

Chapter 2

Twenty Years of Computer Science in Dutch Secondary Education

Computer science (CS) is currently being taught in secondary education all over the world. In the Netherlands, where all students were expected to become computer literate in the lower grades of secondary education (Hulsen et al., 2005) it has been decided not to consider computer literacy as being part of CS. What, then, should be the content of the CS curriculum taught in the higher grades? What should be taught, how and to whom? How should students' achievements be assessed? The answers to these questions completely depend on defining what the objectives of teaching CS are. In the first part of this chapter, these objectives are discussed, along with the content of the Dutch secondary education CS Curriculum when the course was first introduced in 1998, and the experiences resulting from the initial implementation of this curriculum during its first decade, including the setting in which CS found itself.

In the second part of this chapter, we describe the second decade of CS in the Netherlands: CS reached adulthood with established teacher training programs and a new curriculum which is introduced in 2019. Here we describe the events and processes that led to the renewal of the curriculum, the curriculum itself with the principles it is based on and its aims, the current process of teaching material development, the related research, the teacher training, curriculum reform in primary and lower secondary education, and the current situation of CS as an upper secondary school subject, together with the challenges it still faces. From this description of the educational context in the Netherlands also follows the rationale for our research.

This chapter is based on articles Grgurina, N., & Tolboom, J. (2008). The First Decade of Informatics in Dutch High Schools. Informatics in Education, 7(1), 55-74 and Grgurina, N., Tolboom, J., & Barendsen, E. (2018), and The Second Decade of Informatics in Dutch Secondary Education. In International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (pp. 271-282). Springer, Cham.

2.1 The Dutch Educational System

Figure 2 shows the organization of the Dutch Educational System (Jansen, 2007). After completing elementary school at the age of twelve, the students go on to different kinds of secondary education. The VMBO[3] type of school, lasting four years, leads to vocational education. The HAVO[4] type of school (senior secondary education) lasts five years and prepares students for higher professional education, while the VWO[5] type of school (pre-university education) lasts six years and is geared toward further education at a university. Secondary education ends with national exams covering nearly all the subjects taught. In this thesis, we will be focusing solely on senior secondary education and pre-university education. In these schools, every student has the same curriculum in grades seven through nine. While in the ninth grade, a student then chooses the curriculum to be followed in the subsequent higher grades. In 1998, education in these higher grades (10 and 11 for senior secondary education; 10 through 12 for pre-university education) went through major modifications (College voor Toetsen en Examens, 1998). It was decided that the curricula for all existing courses needed to be re-examined and that several new ones should be introduced, one of these being CS. Previously, in 1995, the Course Developer Group had been assigned the task of developing a curriculum for a CS course to be taught in grades ten and higher (Ginjaar-Maas, 1994). In this re-examination, all courses were categorized as either compulsory (e.g. the Dutch language, physical education) or profile courses belonging to one of the four profiles a student can choose from (Culture and Society; Economy and Society; Nature and Health; Nature and Technology). In addition, there were to be elective courses available to all students, one of these being CS. A student first chooses one of the four profiles. Then, in addition to these compulsory and profile courses, every student then takes one or two courses of his/her own choice, either a profile course from another profile or a "free" course.

3 VMBO: Voorbereidend middelbaar beroepsonderwijs: prevocational education
4 HAVO: Hoger algemeen voorbereidend onderwijs: senior secondary education
5 VWO: voorbereidend wetenschappelijk onderwijs: pre-university education

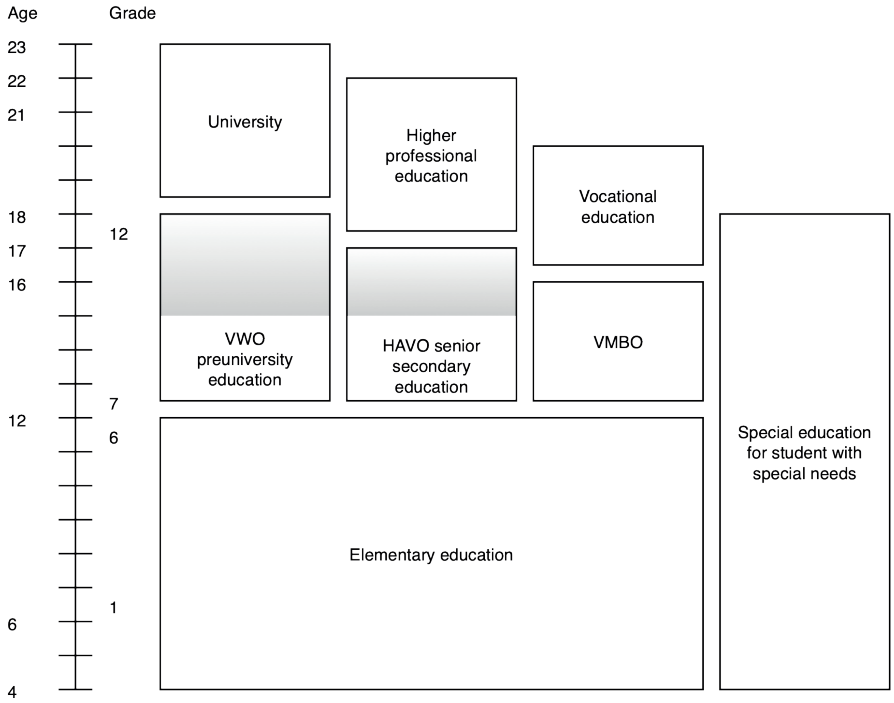


Figure 2: The Dutch educational system. The shaded blocks represent those grades in which the students can choose CS

2.2 The First Decade of Computer Science in Dutch Secondary Education

In the first part of this chapter, we describe the introduction of CS as an elective subject in the higher grades of pre-university and senior secondary education in the Netherlands.

2.2.1 The birth of Computer Science Education in the Netherlands

Before CS was introduced in the higher grades of the secondary education in 1998, only a few schools offered some form of CS education, and in those cases, it was organized by teachers on an individual basis. Only since the Course Developer Group was assigned the task of developing a curriculum for a CS course to be taught in grades ten and higher (Hacquebard et al., 1995), has structural attention been paid to CS in secondary education.

2.2.1.1 Objectives

The curriculum for this new CS course, regarded as a science discipline, was developed with several underlying principles in mind. Its aim was to provide students with an understanding of information technology concepts, and to give them a sense of the potential and limitations of their use in the community as a whole, and, more specifically, of their use in their future careers (Hacquebard et al., 2005). The course was designed to be well within the capabilities of all students, regardless of whether the rest of their curriculum followed the social or the scientific profile. The result produced a course with a multidisciplinary nature, which exemplified how this nature could be applied to complex problems and structures. Furthermore, since CS was not a prerequisite for any subsequent study at the university/college level, there was no need for a national exam; all assessment was to take place in the school itself. These considerations led to the following general objectives:

The CS course at the [...] secondary education level would be focused on providing students with:

- a view of CS and IT, and the relationship between these fields and other subject areas, as well as how they related to technology and society as a whole
- a picture of the role CS and IT would play in their education and career
- hands-on experience with CS and IT through:
  - learning the basic concepts and skills of the subject
  - studying CS problems
  - studying the structures of data processing systems
  - working on system development in groups
- and all this within the context of how CS could be applied in society as a whole (Hacquebard et al., 1995).

2.2.1.2 The Position of Computer Science in the High School Curriculum

CS is not a compulsory course as every school can decide whether to offer it not, nor does choosing to take it depend on any other courses in a student's curriculum. Since its introduction in 1998, it has consisted of 240 study hours for senior secondary education students, and 280 study hours for pre-university education students. These study hours include all the time spent on learning in the classroom, as well as elsewhere. CS course is designed to be taught no earlier than

the tenth grade; that said, schools are free to decide in what grade(s) it should be taught[6].

2.2.1.3 The Computer Science Curriculum

All these considerations resulted in a curriculum that drew its inspiration from the 1994 UNESCO/IFIP curriculum (Weert & Tinsley, 1994); it was recommended that this curriculum cover four themes:

- Theme A: CS in perspective: CS should be examined from several vantage points (science and technology, society as a whole, education and career perspectives, and, finally, from a personal perspective); the result should then provide a student with a general overview. This theme was not intended to be taught on its own, but as an integral part of other themes.
- Theme B: Terminology and skills: in order to be able to develop CS skills, a student needs to acquire adequate knowledge and skills pertaining to hardware, software, organization, as well as to data and information and communication.
- Theme C: Systems and their structures concerns general information issues, various types of data processing systems, and the situations where these are normally used. It covers system theory, computer systems, real-life applications, information systems and new developments.
- Theme D: Usage in a context takes a look at practice. The study of system development and project management, including their social aspects, deals with the relationships between an "information issue" on the one hand, and the development and implementation of IT applications at all kinds of institutions, enterprises and application areas, on the other. This theme is all about letting the students themselves work with CS and IT, and encouraging the intertwining of their CS knowledge and skills with those skills acquired in other subjects in their curriculum (Hacquebard et al., 1995).

6 The secondary school where the first author taught computer science at the time, interpreted this as follows: in senior secondary education there were two weekly 45-minute lessons in the tenth and eleventh grades; in pre-university education two weekly 45-minute lessons in the eleventh grade and three in the twelfth grade

Taking these recommendations into account, the four themes were broken down into 53 specific terms, which would then comprise the CS curriculum[7] (College voor Toetsen en Examens, 1998). A list of these terms is not included in this thesis because their description is quite extensive and substantial, but, more importantly, because they were in the process of becoming obsolete. They were being condensed into a shorter list of eighteen terms that comprised the curriculum for students entering the tenth grade from the fall of 2007 onwards (Schmidt, 2006). Here we give a short overview of the four themes and the eighteen terms comprising the curriculum. The full description of the 2007 curriculum is listed in appendix A.

2.2.1.4 Computer Science Curriculum HAVO/VWO (senior secondary education/ pre-university education)

Theme A: Computer science in perspective
A1: Science and Technology
A2: Society
A3: Study and Career
A4: The Individual

Theme B: Terminology and skills
B1: Data representation in a computer
B2: Hardware
B3: Software
B4: Organizations

Theme C: Systems and their structures
C1: Communication and Networks
C2: Operating Systems
C3: Systems in Practice
C4: Development of Information Systems
C5: Information Flow
C6: Information Analysis

7 The 1998 curricula for the two types of secondary school mentioned in this chapter, as far as the content is concerned, differed only in the details of a small number of terms. However, it was suggested that in the HAVO type of school the emphasis should be placed on practical work, while in the VWO type of school the approach should be more abstract and theoretical (Stuurgroep Profiel Tweede Fase Voortgezet Onderwijs, 1995)

C7: Relational Databases
C8: Human-Computer Interaction
C9: System Development Lifecycle

Theme D: Usage in a context
D. The student should be familiar with the methods and procedures of project management, as well as the project aspects of system development (Schmidt, 2006).

It has been suggested that the 2007 curriculum be implemented in the form of a number of core modules that are the same for both senior secondary education and pre-university education, along with a number of distinct enrichment modules (Schmidt, 2006).

It is interesting to note that, although programming (the practical translation of "algorithmics") is considered a dominant theme in CS by the CS community worldwide (Gal-Ezer et al., 1995), this is not reflected in the Dutch curriculum. Even in the best-case scenario, less than one quarter of the time available is supposed to be dedicated to programming.

2.2.1.5 Teaching Computer Science in the Classroom

In 1998 the major modifications made to education in the upper grades of secondary education came with recommendations for organizing classroom work in a different manner. The students were to be given more freedom and responsibility for their own learning process. Furthermore, obtaining factual knowledge was no longer to be the sole objective of attending school. Acquiring skills and competences became an objective as well (Ginjaar-Maas, 1994). This approach to teaching was a good fit for CS. Students, it was suggested, should spend a lot of time doing practical work and working on projects, mainly solving CS problems. IT applications, in fact, are found in a wide range of areas in "society," and they are always about processing, organizing and communicating large amounts of relevant data. Moreover, IT problems in practice are so intricate and extensive that one person cannot solve them alone, even when the issues in question are relatively simple. Keeping in mind that CS education was not primarily meant to mirror professional practice, these five starting points for classroom teaching were suggested:

CS education should be about students:

1. *Learning about the field of CS* through the acquisition of factual knowledge and skills relative to the ways of thinking and working methods found within CS
2. *Learning to apply CS* through solving CS problems, using the CS knowledge and skills acquired
3. *Learning to deal with interdisciplinary problems:* learning to use CS knowledge and skills in an interdisciplinary context
4. *Learning to cooperate:* learning to practice CS in a structured collaborative way
5. *Learning to reflect*: learning how to learn from the previous four points independently [...] (Ginjaar-Maas, 1994).

2.2.1.6 Assessment

These points of view are reflected in the way the curriculum prescribes assessment and this in turn influences the way CS is taught in the classroom. There is no national exam and all assessments take place at the school level in the form of a so-called *school exam*. A student's CS school exam is a portfolio containing the following parts[8]:

A. Written examination.
B. Practical assignments. The student is to do practical work and come up with a result. When relevant, the process itself is taken into account by giving credit for the documentation describing the processes involved.
C. Project: system development. This is a larger practical assignment to be carried out in groups of at least three students. Each student is expected to work for approximately sixty hours on this project.
D. Activities. Taking part in activities intended to provide a picture of the educational and career perspectives in which IT plays an essential role.

The first version of the 1998 curriculum explicitly states that Part A should contribute forty percent towards the final grade, Parts B and C thirty percent each, while Part D only needs to be covered up to a satisfactory level (College voor Toetsen en Examens, 1998). Soon, however, this was all to change and nowadays the assessment is as follows: Part D, activities, has been removed. Part C, project, has become optional. And Part A, written examination, now contributes to up

8 For senior secondary education, see http://www.eindexamen.nl/9336000/1/j9vvgodkvkzp4d4/vg41h1jtpgy4/f=/bestand.doc (retrieved August 2007) For pre-university education, see http://www.eindexamen.nl/9336000/1/j9vvgodkvkzp4d4/vg41h1jtpgy5/f=/bestand.doc (retrieved August 2007)

to fifty percent of the final grade (College voor Toetsen en Examens, 1998). The practical nature that the course in CS was intended to have is thus emphasized once again by prescribing that practical assignments and/or a project should contribute to at least fifty percent of the final grade.

For the 2007 curriculum it was suggested that the assessment should contain the following parts:

A. Written examination.
B. Practical assignments.
C. Project.

Part A should contribute at least ten percent and at most fifty percent towards the final grade, and part B/C at least fifty percent and at most ninety percent. The grade for part B/C is the arithmetic mean of the grades for parts B and C (Schmidt, 2006).

2.2.1.7 Teacher Training

Prior to the 1998 modifications to the educational system in the Netherlands, there was a small number of teachers teaching CS at their own initiative. These lessons were optional and consisted mostly of programming activities. The teachers developing these activities were usually mathematics or science teachers (Tolboom, 1999). There was no formal CS teacher training. When the decision was taken to introduce CS in the upper grades of senior secondary education and pre-university education, there were a lot of concerns. One of the most urgent was that there were no teachers to teach this course. In considerable haste, CODI[9], a consortium of 12 universities and universities of applied science, was set up to join forces in training teachers, who were to be responsible for the implementation of the CS course in their schools. In the period between 1998 and 2005, those schools that planned to introduce CS sent a teacher each for in-service training. Whereas elsewhere prospective CS teachers were expected to possess extensive knowledge of the subject (Gal-Ezer, 1995), these teachers were by no means required to have any prior knowledge of CS and were only expected to be computer literate at a satisfactory level. In order to obtain their teaching license for CS, they had to complete a two-year program of about 45 ECTS. This program consisted of the following parts: (MinOC&W, 1998).

9 CODI is the Dutch acronym for the Computer Science Teacher Education Consortium (in Dutch: Consortium Omscholing Docenten Informatica).

Course	ECTS
Orientation on CS	3,5
Computer Architecture and Operating Systems	0,7
Visual Programming with Java	5,7
Information Systems: Modeling and Specifying	5
Databases	0,7
Telematics	3,5
Software Engineering	5
Human-Machine Interaction	1,4
Programming Paradigms and Methods of Information System Development	1,4
Didactics of CS	5,7
CS Projects	2,8
Practical Teaching Assignment	10

Table 1: CODI program

For a description of this curriculum from the perspective of the practice of teaching, see Dirks and Tolboom (2000). In 2005, CODI was dismantled leaving a void since no other way was set up to train and license CS teachers. As of 2006, five universities in the Netherlands offer secondary school CS teacher education as a Master's degree program.

2.2.2 The First Decade of Teaching Computer Science

With all these various objectives, points of view, considerations, recommendations, as well as the curriculum itself in mind, one question arises: how does this all work in practice?

CS education has been monitored from the very outset in 1998, and there have been five detailed reports describing the resulting state of affairs (Hartsuijker et al., 2003; Hartsuijker, Kuipers, et al., 2001; Hartsuijker, M.A.G, et al., 2001; Hartsuijker & Westland, 2004; Schmidt, 2007). Other than these reports, there has been no significant scientific research into any of the aspects of CS education in secondary education in the Netherlands.

In the Netherlands, not all schools offer the CS course. During the period 2002-2006, out of about 474 independent schools, the percentage of schools that do offer CS has remained fairly constant at around sixty percent. There are indications that since 2007, this percentage has been rising (Schmidt, 2007). During the CODI era, 369 candidate CS teachers began their studies, and 336 (91%) graduated (Zwaneveld et al., 2007).

During the period 2000 – 2006, the number of secondary education graduates with CS has been on the decrease and seemed to be stabilizing at around ten percent (of the total student population) (Schmidt, 2007).

Setting aside the start-up difficulties in the beginning, a clear picture of the problems encountered and of the accomplishments realized has emerged. From the teachers' point of view, the curriculum was ostensibly too broad and extensive (in terms of available teaching time), forcing them to skip parts of it. Concerning this same issue, they experienced difficulty in judging the amount of attention and time to be given to particular terms. The three textbooks[10] (Bergervoet et al., 2001; Meijer et al., 2001; Van der Laan et al., 2001) on the market did not help much in solving this problem, since each of them had different approaches to the subject matter. Therefore, many teachers were forced to resort to writing their own teaching material (much of which has now been made available to other teachers through the online community on www.informaticavo.nl). This situation, however, was not just perceived as a problem; it was also seen as an opportunity to pay more attention to the subject matter that students and/or teachers themselves found interesting. With this in mind, many teachers were happy that there was no national exam putting pressure on them to work out minutely all of the 53 curriculum terms in the class. In a country where we are used to national exams that ensure the level of students' accomplishments and then serve as gateways to higher education, this has caused quite understandable concerns. What guarantees are there that students attending different schools will end up acquiring a similar body of knowledge? At the moment, since CS is not a prerequisite for any study in higher education, this does not really matter. However, there is an occasional discussion about whether there should be a national exam for CS. The details of this discussion will be described in the section 2.2.3, Discussions.

Bearing in mind that virtually all of the CS teachers come from a non-CS background, it is not surprising to see differing interpretations of the curriculum. In some cases, CS in the classroom has ended up being treated as an exact science, and in some cases the emphasis is put on the use of particular software applications, none of which is in line with curriculum objectives. The picture that students, their parents, and even guidance counselors and other school officials had about CS was often limited and did not fit the broad perspective it was supposed to provide. The experiences of the authors of the original paper (Grgurina & Tolboom, 2008) are

10 These textbooks all consisted of several separate volumes and were accompanied by CD - ROMs and dedicated websites.

a good example: while teaching CS in the tenth grade, they encountered students who expected to spend the lessons doing "computering," by which they meant doing unspecifiable stuff, with the Internet playing a major role in whatever this was. When asked what they expected to learn in CS, they could not formulate a clear answer. And then there were those students who already knew so much that they didn't even consider taking CS because they thought they already knew it all. A solution to this problem was sought by providing instructive lectures (Http://Www.informaticavo.nl/scripts/voorlichting.php.2006) to ninth-grade students about the CS course right at the very moment when they were set to decide which courses to take in the higher grades.

Another difficulty encountered came out of the very foundation upon which the whole CS course was built. On the one hand, this course was meant to be accessible to all students without any prerequisite. On the other, however, the students were supposed to acquire an overview of and hands-on experience with all aspects of CS. When it came to programming, for example, many students had difficulties with complex programming languages such as Java, as had many of their teachers without a CS background when they had been faced with learning this programming language during the CODI training (Tolboom, 1999). Information modeling using a CASE tool for FCO-IM[11] (Bakema et al., 2002) was another stumbling block where many students went astray. Obviously, then, occurrences such as these have played a part in making teachers choose their own interpretations of the curriculum.

Looking on the bright side, both teachers and students appreciated this practical approach to teaching. The practical assignments were highly motivating, and proved to be very valuable by encouraging students to cooperate with each other and take responsibility for their own achievements. They made differentiation among the students possible. They were also illustrative of the practical nature of CS. The way CS is taught in the classroom is a fine example of the new didactic approach behind the modified secondary education system introduced in 1998.

One can get an impression of what is going on in classrooms all over the country by taking a look at the quite lively online community on www.informaticavo.nl. The growing diversity of topics found there is remarkable. A quick look at the collection of tests on programming submitted to the site, for example, shows that the subject is apparently being taught using Visual Basic, Logo, NQC for Lego Mindstorms, Java, Gamemaker and Delphi. The practical assignments submitted

11 FCO-IM stands for Fully Communication-Oriented Information Modelling

also show large variations as well, for example, Build a Company Website; Make a Database for the Administration of a Football Club; Build a User-Friendly Interface for an Information System; and from the first author's own classroom: utilize the debating skills acquired in language classes to debate a particular ethical issue concerning IT.

This diversification and growth are not surprising in a discipline undergoing such rapid changes and new developments, and so now is a good time to pose the question: where do we go from here?

2.2.3 Discussions

Concerns about the future of CS education in Dutch secondary education were reflected in a number of discussions that were going on at that moment, in the first decade of this millennium. One of the hot issues was the question of whether or not to introduce a national exam. In the 2007 report on the implementation of the CS course (Schmidt, 2007), a survey of CS teachers reported the following results:

Answer	Score
As far as I am concerned, a national exam is out of the question	29.2%
As far as I am concerned, a national exam is only to be considered under strict conditions	26.1%
I am not an advocate of a national exam, but I'm not against it either	16.9%
I can see the advantages of a national exam	13.8%
I truly believe in a national exam	13.8%

Table 2: CS teachers' survey results on the question of a national exam

When asked about their reasons, the great majority of those opposing answered that they feared losing the freedom to design the content of the subject they taught, followed by the argument that CS was not a prerequisite for any subsequent study at the higher-education level. The advocates of a national exam posited that it might help strengthen the position of this school subject in the curriculum alongside all the other school subjects. They also felt that it would lessen the differences in the levels of accomplishment found in students. As a result, higher education would have a better picture of what to expect from first-year students who had taken CS during secondary education. The advocates assumed that only about sixty percent of the subject matter needed to be examined by means of a national exam, because only a limited number of curriculum terms were suitable for practical examination. Furthermore, aspects of the curriculum, such as cooperation, task-

sharing and working on projects, were difficult to examine in a national exam. Therefore, a national exam could, just as in the case of many other school subjects, contain about 60% of the curriculum (Schmidt, 2007).

CS teachers were obviously not in favor of a national exam, but they were not the only party involved in this discussion. Universities did advocate one. National politics, with its tendency to want to exercise a controlling influence on the output quality of secondary education, saw a national exam as a suitable instrument to do just that.

Still another discussion surrounding CS was the question of whether to make CS compulsory. A slight majority of CS teachers thought that CS should be compulsory in the lower grades of secondary education so that students would have a better picture of this subject when they choose their profile and the curriculum to follow from the tenth grade on. CS in the higher grades would then be able to consist of several modules, since the students would already have a foundation to build on. In addition, this would create a continuous trajectory from elementary school all the way through to the higher grades of secondary education. About one quarter of the teachers, however, were opposed to this idea. Other scenarios, such as a common core course for all students, with separate modules for each profile, or even a distinct CS course for each of the profiles, can count on roughly equal numbers of advocates and opponents, or just simply a majority of opponents.

As of the fall of 2007, education in the higher grades of secondary education in the Netherlands has once again been subjected to certain modifications. The leitmotif was that schools should be granted more autonomy and choice in the way they organize education, and so make the whole of education more manageable for schools through, among other things, streamlining the amount of time each course is allocated. Again, these modifications were favorable for CS course in general and for individual schools in particular. To begin with, the hours allocated to the whole of the CS course increased to 320 for senior secondary education and 440 for pre-university education. Furthermore, the curricula for all the subjects have been simplified so as to decrease the number of terms, and the terms themselves are no longer described in such great detail. For CS this means that, following recommendations based on classroom experience (Hartsuijker & Westland, 2004), no new terms have been added. The result was that the new curriculum consisted of the eighteen terms listed in section 2.2.1.4.

There was more good news for CS. Not only were schools being given total freedom in designing their assessment procedures, they were also being encouraged to teach topics that extend, deepen, and go beyond the curriculum terms (Tweede Fase Adviespunt, 2006).

And last but not least, there is news from the research side. The Technical University of Eindhoven (TUE) and the Open University of the Netherlands have begun a research project to analyze and summarize the relevant research literature for pedagogical content knowledge (PCK) for CS teachers and to link this PCK to practice in the Dutch classroom. The aim is to create an inventory of research-based best-practice characteristics.

In spite of all this good news for the CS course in general, the prospects for a bright future were hampered by various problems.

The publishers of two of the three textbooks (Bergervoet et al., 2001; Meijer et al., 2001) have decided to cease publication of their textbooks. The authors of these textbooks took steps to continue development of the teaching materials and to make them available online. There were university-based projects aimed at developing teaching materials as well. However, together these initiatives did not provide enough teaching material to meet the needs of the increased number of study hours, leaving the teachers to their own devices once again.

There were many non-licensed teachers teaching CS –—an estimated two out of five — the same as was reported in Israel back in the 1990's (Gal-Ezer, 1995), and this situation did not look like it was going to be changing any time soon (Schmidt, 2007). Despite the fact that since the fall of 2006 there were five universities in the Netherlands — Utrecht University, University of Groningen, University of Twente, Delft University of Technology and Eindhoven University of Technology— where one could become a licensed secondary education CS teacher, the numbers of students did not nearly match the demand from schools. The reasons for this were numerous and complex. In order to become a licensed CS teacher, as a rule one needs to have a Bachelor's degree in CS, and then follow a Master's in Education and Communication. Almost none of the CS Bachelor's students took this route because a career in education was often perceived as being of low social status, coupled with low pay and presenting almost no career prospects, while the booming economy had so much more to offer. On top of all that, a typical school did not have enough weekly CS lessons scheduled to offer full-time employment to a CS teacher. On the other hand, a lot of the people who did want to become licensed CS teachers had an IT background and/or were

licensed to teach other subjects, but did not qualify for the Master of Education and Communication due to the lack of a formal CS Bachelor's degree. Needless to say, this paradoxical situation did no good for CS education as a whole.

Another problem, probably the toughest of them all, and experienced internationally as well (Downes, 2007), is the perception of CS held by the whole of the population, with education policy makers regrettably being no exception. Two examples illustrate this unfortunate situation:

The Ministry of Education was looking ahead and considering the future of educational innovations and reforms, and to this end it set up an advisory board and asked it to formulate a vision for future educational developments. This board consisted of 33 members, mostly university professors, several secondary education teachers, along with a few policymakers and students. None of them had any CS background, which, in our opinion, is regrettable, because the advice the board presented to the Ministry paid scarcely any attention to CS education, barely mentioning it at all. The 232-page document mentioned mathematics 263 times, and CS 10 times (Society, 2005).

Another example was even more serious in our opinion. One of the new courses introduced as part of the educational modifications in 2007 is NLT (Nature, Life and Technology). It is meant to be a cross-subject science course offered to those students choosing one of the Nature profiles (see section 2.1). Even though the proposed NLT curriculum[12] contained terms pertaining to IT and bioinformatics, CS teachers were not licensed to teach it, while the teachers of mathematics, physics, chemistry, biology and geography were.

2.2.4 Concluding Remarks about the First Decade of Computer Science

During the first decade of teaching CS in Dutch secondary education, the objectives outlined in the 1990s do seem to have been achieved. What CS education was going to look like in its second decade depended on the outcome of the discussions about the introduction of a national exam and whether to make it a compulsory subject, as well as on the repercussions from the fact that many CS teachers were not licensed and/or adequately trained. Furthermore, it was not yet clear whether the government intended to reform education in the upper grades of secondary education again, and if so, what consequences this will have for CS education. And, last but not least, we believe that clearing up the misconceptions surrounding CS and bringing proper attention to bear on its significance would contribute to a bright(er) future for CS education.

12 For a description of the NLT curriculum, see http://www.betavak-nlt.nl.

2.3 The Second Decade of Computer Science in Dutch Secondary Education

In this second part of the chapter, we report on the second decade of the CS course in Dutch secondary education: it did not become compulsory and no national exam was introduced. However, there was a CS curriculum reform, new technologies have changed our lives in ways unforeseen ten years earlier and these developments have led to a whole new set of discussions and challenges surrounding CS.

Here we focus on the events that led to the curriculum reform, the curriculum itself, the current situation of the CS course, and on other developments related to CS education such as an advised introduction of a foundational module in lower secondary education (KNAW, 2012) focusing on digital literacy. In the Netherlands, digital literacy is considered to consist of four skills: basic ICT-skills, media literacy, information literacy and computational thinking (Thijs et al., 2014b), as described in section 1.2.

2.3.1 Situation in Practice

In this section, we describe the present situation of the CS course in secondary education in the Netherlands. Beginning with the most recent figures from 2017, we present the results of the 2014 research project charting the actual situation in schools, describe the events leading to the new 2019 curriculum, and finally, we discuss the newest developments.

2.3.1.1 Schools, Students and Teachers

Looking at the numbers of schools offering the CS course and numbers of students following it, we see that during the period 2002-2006, out of about 474 independent schools, the percentage of schools that do offer CS was fairly constant at around sixty percent with about ten percent of students following it (Schmidt, 2007), cited in (Grgurina & Tolboom, 2008). In the period 2011-2017 out of approximately 500 schools, the percentage of schools offering CS dropped from 55% to 47%, while the proportion of students following it remained fairly constant at around 11% for HAVO and around 12% for the VWO type of school (DUO, 2018).

In 2013, the government commissioned an inquiry and a report by the Netherlands Institute for Curriculum Development (in Dutch: Stichting leerplanontwikkeling, SLO) to explore the teachers' ideas about the necessity to

change the CS curriculum. In this section we present the data that were collected concerning the teachers and their ideas about desired curriculum changes. and in the section on the curriculum (section 2.3.2), we report extensively about the results of this inquiry (Tolboom et al., 2014). Regarding the teachers themselves, 89% are male and 11% are female. Almost two out of three are the sole CS teachers in their schools, with 31% of schools having two teachers and 4% having three teachers. The majority — 55% — teaches another subject as well, both in science (mostly mathematics) and in humanities and other subjects (e.g. geography, history, economy, arts and languages). When teaching CS, almost two-thirds of them cooperate with the teachers of other subjects, mostly physics, mathematics and business, but also from other sciences, humanities and, notably, arts. Most of the teachers — 62% — have been teaching CS for more than six years. Concerning education they attended to qualify them to teach CS, 52% of the teachers said they attended the CODI in-service program (see the first part of this chapter), 18% followed a university educational master, and 36% did something else. Since these numbers add up to more than 100%, we suspect that a number of teachers took several educational routes. Most teachers actively engage in staying up-to-date by following in-service training courses (67%), participating in teacher networks (65%), reading professional literature (81%) and engaging in other activities (39%). Still, most of them — 62% — find the in-service training courses offered to be insufficient and see that as a problem threatening the quality of CS as a school subject (Tolboom et al., 2014).

When it comes to teaching materials they use, we see that the online books offered by the three publishers are often combined with each other and with other teaching material, either found elsewhere or written by the teachers themselves (Tolboom et al., 2014). Competitions have made their way into the classroom too: The CS Olympiad[13] including CodeCup[14] and the international Bebras competition (Dagienė & Stupuriene, 2016a).

When asked about their opinion on the CS curriculum, 7% of the teachers said they were not familiar with it. One out of four teachers did not find it useful, 38% would like it to contain less or other learning objectives, and 36% were satisfied with it. When asked what learning objectives to strike from the curriculum, almost half of the teachers — 45% — said none but added that they would like the curriculum to offer more guidance. Other teachers, those wishing to strike some learning objectives, most often mentioned those related to business aspects

13 http://www.informaticaolympiade.nl
14 http://www.codecup.nl/intro.php

of CS (organizations and information flow) and information analysis. When asked what they were missing in the curriculum, they most often mentioned (more) programming, social and professional aspect of CS, security aspects, and networks and communication (Tolboom et al., 2014).

2.3.1.2 Teacher Training

Eight years after the introduction of the CS course in secondary education, regular pre-service teacher training was established first at the University of Groningen in 2006, and then another four universities followed. As pre-service university teacher training for other subjects, this is a two-year educational master. Specifically, for CS, only students with a university bachelor degree for CS (or an equivalent) are admitted. During this master, the students both deepen their subject matter knowledge and learn about teaching through extensive internships at school and accompanying theoretical underpinnings concerning CS didactics, pedagogy and educational science (Barendsen & Tolboom, 2016). For those who already possess a master's degree in CS, a one-year program is available. The number of students obtaining this degree and thus becoming qualified CS teachers in the years 2008-2013 was nation-wide six on average; in 2014 and 2015 there were four, and in 2016 there were three. These numbers do not come anywhere near the perceived need: the expected unfulfilled vacancies are estimated to be 36 in 2018, 52 in 2020 and rising to 86 in 2025 — implying that roughly a third of schools offering CS might not be able to find qualified teachers (Adriaens et al., 2016).

While the number of regular university students interested in becoming teachers is tiny, every year in the meetings of CS teachers' educators from the five universities involved, we hear about dozens of professionals with a background in IT industry who show interest in becoming CS teachers. However, due to the strict requirements, only a few get admitted to the teacher training straight away.

To help people with university degrees in technology or science to meet the teacher training admission requirements, in 2015 the beta4al[15] project was started for chemistry (chem4all), and in 2016 physics (natk4all) and CS projects were added: inf4all.[16] Beta4all project is a result of a cooperation of universities offering teacher training (i.e., educational master) where a number of courses are offered to interested candidates as well as to interested teachers, in the form of specifically tailored courses of 6 ECTS each. For CS, these courses were Foundations,

15 http://www.beta4all.nl
16 http://www.beta4all.nl/prog_inf4all.htm

Algorithms, Advanced Object-Oriented Programming, Networks, Databases and Information Retrieval; Media, Games and User experience; Artificial Intelligence, and finally, Internet of Things. The courses are taught biweekly during one semester in the form of three-hour lectures on Fridays in Utrecht, in the center of the Netherlands, thus allowing people from all over the country to participate.

The admission procedure is then as follows: a candidate interested in becoming an CS teacher approaches the university of their choice where their educational background and relevant professional experience are assessed. In case the candidate is not admissible yet, a tailored plan is put together consisting of appropriate inf4all courses (and possibly other courses taught at the university itself) to be taken.

2.3.2 Curriculum Reform and Curriculum

Here we describe the events leading to the CS curriculum reform and the new curriculum itself.

2.3.2.1 The Royal Netherlands Academy of Arts and Sciences Report (KNAW report)

Ever since the late 2000's, in the Netherlands, just like in the international field of experts (Gander et al., 2013), several stakeholders have been expressing concerns about outdated curriculum and position of CS as a school subject in general and advocated a curriculum revision. However, the government refused to draw consequences from periodic evaluations (Schmidt, 2007) and the Ministry of Education, Culture and Science maintained that there was no apparent need for a curriculum reform since there were no complaints "from the field". In 2012, triggered by serious concerns expressed by a number of influential CS education specialists, The Royal Netherlands Academy of Arts and Sciences (in Dutch: Koninklijke Nederlandse Akademie van Wetenschappen, KNAW) formed a committee to investigate the situation of CS in secondary education. This committee wrote a critical report containing five recommendations aimed at improving CS education in general, reaching far beyond the scope of the current CS course. The report recommends to "Completely overhaul the optional subject CS in the upper years of HAVO and VWO" and suggests to make it modular and flexible, relevant and attractive to all students. Furthermore, it recommends to "Introduce a new compulsory subject Information & communication in the lower years of HAVO and VWO" and goes on to make recommendations about

encouraging "interaction between these subjects and other school subjects", adequately training teachers, instructing higher education to collaborate in this regard and, finally; to "promote instruction in digital literacy" to help achieve the goals set in the nation's ICT policy concerning innovation and economic development (KNAW, 2012).

Even though this report was received with great enthusiasm by the CS field, the government was still reluctant to initiate a curriculum reform

2.3.2.2 The Netherlands Institute for Curriculum Development Report

In 2013, under pressure from the stakeholders, the government commissioned an inquiry and a report by the Netherlands Institute for Curriculum Development (in Dutch: Stichting Leerplanontwikkeling, SLO) in order to assess (1) what is needed to realize a modern and attractive CS education in upper grades of senior secondary education and pre-university education and (2) in case it turns out that a change of curriculum is required, what should that change entail. The Institute appointed a team of three researchers for this job. First, they conducted a literature study about the importance of CS education, both nationally and internationally. Then they invited CS teachers to fill in an online questionnaire about their current CS teaching practice and about their wishes and suggestions concerning possible changes in the CS curriculum. Subsequently, they conducted in-depth interviews with a small number of these teachers. Finally, they consulted a large number of CS experts, not all of them involved in secondary CS education.

This investigation resulted in early 2014 in a report containing three recommendations and a description of four factors playing a decisive role in subsistence of CS in secondary education. First of all, the report recommends to design a new CS curriculum aimed at a diverse student population, varied enough to be relevant and attractive to all students. The second recommendation instructs to design a curriculum containing a limited number of compulsory learning objectives and a number of objectives from which a student can choose. Finally, it recommends to keep the assessment as it is (i.e. at the school level only, rather that introducing a national final exam which most other subject have). Furthermore, the report lists four critical factors which need to be addressed in order to make and keep CS a viable school subject:

1. quality of the assessment (with no national final exams, there is no quality control across different schools);
2. development of modular teaching material in order to provide for rapid advances of the discipline;

3. in-service training of the teachers;
4. training of adequate numbers of new teachers (Tolboom et al., 2014).

Within weeks of the publication of this report, the government appointed a committee of nine members — teachers, CS specialists, experts from higher education, and curriculum and assessment specialists —to redesign the CS curricula for the HAVO and VWO types of schools, that was formulated as follows:

- The committee's task is to design a new curriculum for the elective course CS in the upper grades of HAVO and VWO types of school.
- The purpose of the new curricula is to enhance the quality of this course by updating and modernizing the its learning objectives.
- The curricula need to be formulated globally: for the teachers it needs to be clear what the curricular goals are, while at the same time the schools keep sufficient room for their own interpretation.
- There is sufficient distinction between the HAVO and VWO curricula without them being two separate curricula.
- Each curriculum contains a number of compulsory core components and elective components. The elective components are related to the educational tracks the students follow (either one of the two humanities tracks or one of the two science tracks), yet they are within reach of all the students choosing CS, regardless of the track they actually follow.
- The curriculum design is to follow the context-concept approach.

Furthermore, when formulating the new curriculum, the committee needs to take into account the following requirements: The assessment should remain as it is, consisting of a school exam only and no national exam. This curriculum is to be aligned with the curricula in lower secondary and primary education, which are to be developed in the near future. The study load needs to remain the same. The curriculum should not be overloaded and it must be possible to implement it within the available time. The curriculum does not favor any particular didactical approach; it describes the "what" and not the "how". The new curriculum needs to be able to count on the wide support of the teacher community. The textbook publishers need to be kept informed on the progress in order to enable them to prepare the teaching materials in time (*Opdracht vernieuwingscommissie informatica 2014-2015*, 2014).

2.3.2.3 Lorentz Workshop

At the same time when The Netherlands Institute for Curriculum Development inquiry took place in 2013, but independently of it, a number of leading Dutch scholars proposed to organize a Lorentz Workshop.[17] By the time the workshop took place in September 2014, the committee for the reform of the CS curriculum had already been appointed.

The aim and goal of the Lorentz workshop were described as follows: "secondary education on CS and digital literacy urgently needs thorough improvement. The workshop intends to develop a contemporary design for the discipline, following and learning from similar efforts in other countries." The attendees were international experts from Belgium, France, Germany, Israel, Lithuania, UK and USA and Dutch computer scientists, teachers, education specialists, students and policy makers. During the five-day workshop, the international experts presented their country reports and various topics were discussed in focus groups:

- Definition: what comprises an ideal curriculum for a "digital literacy" course in the lower grades of secondary education and an "CS" course in the upper grades;
- Sustainability: how to make "the curriculum sustainable in a rapidly developing field";
- Concepts and contexts: context-based teaching approach similar to the one adopted for other science subjects;
- Diversity: catering to the needs of students with different educational backgrounds and interests;
- Integration with other subjects in secondary education;
- Teacher training (Center for Scientific Workshops in All Disciplines - Computing in Secondary Education, 2014).

On the final day, the results of the workshop were presented and discussed in a meeting with a representative of the Ministry of Economic affairs, a member of parliament specialized in education, a representative of the Dutch CS teacher association,[18] representatives from higher professional education and universities, a CEO from industry and the chairman of the KNAW committee who authored the KNAW report (2012).

17 "The Lorentz Center is an international center that coordinates and hosts workshops in the sciences, based on the philosophy that science thrives on interaction between creative researchers." http://www.lorentzcenter.nl/aim.php

18 http://ieni.org

2.3.3 The New Computer Science Curriculum

2.3.3.4 Design Principles

The new curriculum is based on a number of design principles intended to make it modern and robust, and to cater to the needs of all those involved in its use. To ensure the relevance of the new curriculum in the long term, the curriculum committee decided to follow the so-called concept-context approach — a pedagogical principle that was already applied to several science subjects, for example chemistry (Bennett & Holman, 2002). The fundamental concepts — that were described concretely — were separated from the contexts described generically. In order to deal with the diversity of students, stemming from their varying interest in CS, the educational track they follow (science or humanities), and the fact that division into HAVO and VWO type of school is not always reflected in students' achievements for CS, it was decided to divide the curriculum into a core curriculum that is mandatory for all students taking the CS course and a number of elective themes. Furthermore, as many see CS as a constructive discipline where one engages in creation of artifacts, 'design and development' is positioned as a central skill in the new curriculum. Finally, in order to balance guidance and freedom experienced by the CS teachers, the committee drafted comprehensive learning objectives: 30 of these are in the core curriculum and the other 34 in the elective themes, thus allowing the schools to shape their CS educations according to their preferences (Barendsen et al., 2016; Barendsen & Tolboom, 2016).

2.3.3.5 Learning Objectives

The learning objectives of the new curriculum are organized in six compulsory domains forming the core curriculum and twelve elective themes from which a HAVO student needs to choose two and a VWO student four. The domains forming the core curriculum are: (A) Skills, (B) Foundations, (C) Information, (D) Programming, (E) Architecture, and (F) Interaction. The elective themes are: (G) Algorithms, computability and logic, (H) Databases, (I) Cognitive computing, (J) Programming paradigms, (K) Computer architecture, (L) Networks, (M) Physical computing, (N) Security, (O) Usability, (P) User Experience, (Q) Social and individual impact of CS, and (R) Computational Science (Barendsen & Tolboom, 2016). The full text of the curriculum is listed in appendix B.

2.3.4 Teaching Materials for Elective Themes

In this section we describe the project in which teacher teams develop teaching materials for the elective themes. We first provide the general description of this project and then focus on one particular team — the one working on Computational Science.

2.3.4.1 Teacher Teams Developing Teaching Materials

The curriculum specifies only high-level learning objectives and does not provide further details about them, nor about the instruction or assessment. In line with the Dutch tradition, this is left to the educators and authors of teaching materials, usually employed by publishing companies. In the Netherlands, there are three publishers of teaching materials for the CS course. With 11 to 12 percent of the students in HAVO and VWO schools electing to take this course (DUO, 2018), the market for the publishers is rather small. This situation, combined with the fact that elective themes in the new curriculum will inevitably be chosen by even smaller numbers of students, means that the publishers have no financial incentive to develop teaching materials for the elective themes and are only interested in developing teaching materials for the core domains. To alleviate this problem, the Ministry of education provided financial means and asked the Netherlands Institute for Curriculum Development (SLO) to coordinate a project where teams of teachers would develop teaching materials for elective themes. SLO developed a procedure describing the participants and stakeholders in the project, the guidelines outlining the process they engage in, and finally, the products to be delivered. In accordance with this procedure, for each of the twelve elective themes a team should be formed, consisting of at least two CS teachers, an expert and a teacher educator specialized in didactics of CS. First, the team writes a global description of the module they work on, which specifies the intended learning outcomes, target audience, planning and other relevant details. Then they engage in the actual writing of the module which needs to satisfy the following criteria:

- suitable for self-study because not all teachers are expected to possess adequate expertise for that particular domain
- embed the intended learning outcomes in rich and relevant contexts
- incorporate at least one of the three basic skills in the curriculum, namely: design and development, using CS as a perspective, and finally, cooperation and interdisciplinarity

- suitable for both the HAVO and the VWO students, yet provide for their differences
- suitable to be published online
- accompanied by teachers' instruction and a suitable form of assessment (e.g. a test or a practical assignment).

When a module is finished, it should be tested in at least two schools. The feedback from the teachers and their students who engage in testing of a module should then be collected, and a new version of a module should be written. The final version of the module should be presented to an external expert and a certifying body for final approval. This certification serves as quality control in multiple ways, not the least to partly compensate the lack of a national exam and corresponding lack of quality control and lack of ways to compare students' achievements across different schools.

2.3.4.2 Development of Teaching Materials for Computational Science

This process is illustrated with the example of teaching materials being developed for the elective theme R: Computational Science which is the focal point of this thesis. As a teacher educator specialized in didactics of CS and a researcher, I lead a team developing teaching materials and assessment for Computational Science. The results of the research presented in this thesis — the operationalization of the intended learning outcomes of Computational Science, suggestions for suitable data sources to monitor students' learning outcomes, awareness of students' challenges when engaging in modeling activities (all described in section 3.5), teachers' ideas and suggestions about instruction and assessment as well as our insight into their PCK (described in section 4.3), together with our assessment instrument (described in chapter 5) — supplemented with the decision to employ the 4C-ID instructional design (Kirschner & Merriënboer, 2008), form the starting points for the team and thus insure that the teaching materials, accompanying assessment instruments and teachers' manuals will be set up upon solid theoretical foundations. All aspects of the implementation in schools will be monitored closely and will form the input for further research into the teaching and learning of Computational Science.

2.3.5 Research

During the last decade, CS education research (CSER) in the Netherlands was given a new impulse with the appointment of the first full professor in CS

education. He set up a nation-wide research group conducting research on various aspects of teaching and learning of CS in primary, secondary and tertiary education. The research topics in secondary education include programming, design-based CS education, assessment, context-based teaching and learning of fundamental concepts including algorithms, and finally, Computational Science — the research project described in this thesis. In all of the research projects mentioned here, specific attention is given to the teachers and their pedagogical content knowledge (PCK).

We consider this as a special and a beneficial situation when aiming at research-based CS curriculum development.

2.3.6 Computer Science Curriculum Reform in Primary and Lower Secondary Education

One of the recommendations of the 2013 report by The Royal Netherlands Academy of Arts and Sciences (2012) was to introduce digital literacy into the Dutch lower secondary education. The report led to the chain reaction described earlier in this chapter, thus mainly focused at what already existed in the curriculum: CS in upper secondary education. Nevertheless, both primary and secondary schools started experimenting with integrating digital literacy in their school-based curriculum. Very interesting initiatives emerged, deployed by creative and innovative teachers at primary and secondary schools. While experimenting, the question arose at schools, from teachers, students and parents: is what we are doing now aligned with the formal national curriculum? The answer to this question was: yes, but only because in this formal curriculum — outside upper secondary education — the relevant learning objectives are global and non-specific.

The somehow strange situation — where educators were asking for guidelines that did not exist — was resolved in March 2018, when the curriculum.nu project (mentioned in section 1.2) was started: design teams consisting of selected teachers and school administrators started to rethink the whole of the curriculum. One of the domains to be inspected is digital literacy. A national system for feedback was implemented in order to facilitate revision of and wide support for the vision. This project is planned to end in the fall of 2019 with an advice by the design team on how to revise the Dutch primary and secondary curriculum with respect to digital literacy. This may lead to the introduction of learning goals with respect to digital literacy in primary and lower secondary education.

We see that CS, now clearly visible in the upper secondary education, has gained that much momentum, that it could possibly contribute to reforms in the Dutch educational system.

2.3.7 Conclusion and Discussion

In this chapter, we described the present situation of the elective CS course in upper secondary education in the senior general secondary education (HAVO) and the pre-university education (VWO) in the Netherlands, the CS curriculum reform together with the events leading to it and we sketched the developments related to teaching CS in primary and lower secondary education. We see that on one hand, significant progress has been made with five universities offering regular teacher training (in the fall of 2019 joined by both universities in Amsterdam: the University of Amsterdam and the VU Amsterdam), a nation-wide research group performing research of international relevance and a grassroots movement with parents, teachers, headmasters and other stakeholders demanding more, earlier and broader CS education. On the other hand, the teacher population forms one of the weak spots in the CS ecosystem: many teachers are underqualified, far too few new CS teachers are being trained, and it is not clear who should teach CS or digital literacy if it gets introduced into primary and lower secondary education. Some stakeholders see the CS curriculum reform as a missed opportunity to make CS a mandatory subject or for the introduction of a national exam. Finally, it is not clear yet what will happen with the desire and all the initiatives employed to introduce CS, in whatever form, into the primary and lower secondary education.

However, in 2015, when the contours of the new 2019 CS curriculum became visible and it was clear that it was going to contain the elective theme Computational Science, we decided to embark on this research project to explore the pedagogical aspects of teaching Computational Science in the Computer Science course in secondary education in the Netherlands, and the result is this thesis.

Chapter 3

Defining and Observing Modeling and Simulation in Computer Science

In this chapter, we describe our first study focusing on the Computational Thinking aspect *modeling and simulation*. We conducted a case study analyzing the projects of 12th grade secondary education CS students in which they made models and ran simulations of phenomena from other disciplines. We constructed an analytic framework based on literature about modeling and analyzed students' project documentation, recordings of student groups at work and during presentations, survey results and interviews with individual students. We examined how to discern the elements of our framework in the students' work. Moreover, we determined which data sources are suitable for observing students' learning. Finally, we investigated what difficulties students encounter while working on modeling and simulation projects. Our findings result in an operational definition of the learning objective Computational Science[19] contained in the 2019 Dutch secondary CS curriculum, and provide input for future development of both assessment instruments and instructional strategies.

This chapter is based on the paper Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., & Suhre, C. (2016). Defining and observing modeling and simulation in informatics. In International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (pp. 130-141). Springer, Cham.

19 In this thesis, we use terms *modeling*, *modeling & simulation*, *simulation modeling* and *computational science* interchangeably, unless explicitly stated otherwise.

3.1 Introduction

Following the increasing availability of computers in schools, several initiatives have been employed to support students' learning in various disciplines through the use of computer models (Blikstein & Wilensky, 2009; Pfefferova, 2015; Taub et al., 2014). Caspersen and Nowack (2013b) argue why they „believe understanding and creating models are fundamental skills for all pupils as it can be characterized as the skill that enable us to analyze and understand phenomena as well as design and construct artifacts." Wilensky argues, "Computational modeling has the potential to give students means of expressing and testing explanations of phenomena both in the natural and social worlds" (Wilensky, 2014). Granger claims: „Modeling is the new literacy" (2015). This belief is also expressed in the fact that as of 2019, modeling and simulation (together called Computational Science), will be included in the new Dutch secondary education CS curriculum, described by the following high level learning objectives: "Modeling: The candidate is able to model aspects of a different scientific discipline in computational terms" and "Simulation: The candidate is able to construct models and simulations, and use these for the research of phenomena in that other science field." Modeling itself will be a part of the compulsory core curriculum, described as "Modeling: The candidate is able to use context to analyze a relevant problem, limit this to a manageable problem, translate this into a model, generate and interpret model results, and test and assess the model. The candidate is able to use consistent reasoning." (Barendsen & Tolboom, 2016)

Modeling and simulation can be viewed as aspects of Computational Thinking (CT) (Wing, 2006) as they involve decomposition of open-ended problems and the construction and evaluation of models that simulate the nature of these problems in order to be able to provide solutions to those problems.

Prior to this study, we refined the CSTA definition of CT (Grgurina, 2013), explored teachers' PCK (Grgurina et al., 2014a, 2014b); and made an initial exploration of the computational modeling process (Grgurina et al., 2015).

3.1.1 Aim of the Study

In this study, we focus on CT skills related to modeling and simulation and we explore highly cognitively complex set of students' activities related to modeling, in particular as an aspect of CT rather than as an aspect of e.g. mathematics (Maaß, 2006). Our primary goal is to establish an operational description of the

learning objectives of Computational Science — Magnusson's component M1 (i.e. goals and objectives), and additionally to determine what data sources are suitable to monitor students' learning outcomes when engaging in modeling activities, and finally to explore what specific challenges do the students experience when engaging in modeling activities.

We address the following research questions:

1. How can the intended learning outcomes of Computational Science (modeling and simulation) be described in operational terms?
2. What data sources are suitable to monitor students' learning outcomes when engaging in modeling activities?
3. What specific challenges do the students experience when engaging in modeling activities?

The first question addresses Magnusson's component M1 — goals and objectives. The second question contributes to Magnusson's component M4 — methods of assessment — as we plan to use our findings as input for a later study into a CT assessment instrument (see chapter 5). The third question addresses Magnusson's component M2 — students' understanding as our findings will help to design teaching materials for modeling and simulation and thus indirectly contribute to Magnusson's component M3 — instructional strategies.

3.1.2 Related work

Previous work on characterizing modeling is done mainly in the areas of mathematics and natural sciences; see the following section. Research on making students' learning process and outcomes visible has focused mostly on CT aspects such as algorithmic thinking or programming. The employed assessment instruments range from tests with closed questions (Gouws et al., 2013b), tests with open questions (Meerbaum-Salant et al., 2013; Werner et al., 2012), surveys (Werner et al., 2012), recordings or observations of students at work (Meerbaum-Salant et al., 2013), examination of programming projects (Brennan & Resnick, 2012; Meerbaum-Salant et al., 2013; Werner et al., 2012) to interviews with students (Brennan & Resnick, 2012; Grover, 2011) and teachers (Meerbaum-Salant et al., 2013). In particular, Brennan and Resnick (2012) "are interested in the ways that design-based learning activities […] support the development of computational thinking in young people" and they explore three approaches to assessment of the development of CT of the children engaged in such activities. They discuss strengths and limitations of each of these approaches extensively

and subsequently advocate a comprehensive approach to assessment that utilizes several data sources — an approach that we explore in this study too.

3.1.3 Context of the study

Our exploratory case study was carried out during a project-based lesson series within the CS course in the 12th grade of secondary education (in Dutch: VWO 6) where students studied modeling and simulations. They used NetLogo to program models of phenomena from other disciplines and to explore them through running simulations. During a six-weeks period they studied Modeling and Simulations with NetLogo. The first three weeks were dedicated to studying the instructional materials from a regular textbook (Heuvelink, A. et al., 2008). During the rest of the period, the fourteen students comprising this class worked in seven groups on a practical assignment where they investigated a phenomenon of their choice by making a model in NetLogo and exploring it through running simulations. When necessary, students were assisted in formulating their hypotheses or research questions. The entire process was strictly planned and contained milestones when the students turned in the required project documentation. At the end of the period, each group presented its model to the rest of the class and the students were encouraged to discuss their models, results, design choices, programming issues and other relevant questions. After the presentations, they turned in the final part of the project documentation where they described the feedback they got and their reaction to it. A few days later, twelve students (six groups) who finished their projects, turned in their final reports and NetLogo programs.

3.2 Modeling and Simulation

There is extensive literature on modeling in science and especially in mathematics. We take the latter as starting point and discuss modeling in mathematics first, and then simulation modeling (in section 3.2.2) as a special case of modeling.

3.2.1 Modeling

Van Overveld et al. (2015) distinguish two purposes of modeling: scientific research and technological design, and lists a number of goals that can be obtained through modeling: explanation, prediction, compression, abstraction, unification,

analysis, verification, communication, exploration, decision, optimization, specification, steering and control, and finally, training. The mathematical modeling process can be viewed as a problem-solving activity (cf. (Polya, 2008)). We adopt the operationalization by Van Overveld (2015):

1. Definition stage: the problem is stated and researched in the context domain (this is also considered a core aspect of CT (Wing, 2006)). The purpose of the model is formulated and a study is planned.
2. Conceptualization stage: Data are collected and a conceptual model is constructed and validated. In the process of abstraction, it is decided what details to highlight and what details to ignore.
3. Formalization stage: the conceptual model is transformed into a formal model.
4. Execution stage: the model is being used for its purpose: this means solving the (mathematical) problem.
5. Conclusion stage: the results of the execution stage are analyzed and translated back into the problem domain, involving the presentation and interpretation of the results.

In addition, Perrenet and Zwaneveld (2012) explicitly distinguish between the non-mathematical world containing the definition stage, conceptualization stage and conclusion stage on the one hand, and the mathematical world containing the formalization and execution stages on the other hand. Following each of these stages, reflection needs to take place: to check if any revisions are necessary by repeating that stage, to validate and verify the model, to assess the plausibility of the result and answer the initial purpose, to communicate the results and to learn from what one has done. After the completion of the modeling process, a reflection takes place and the whole process is possibly repeated. Hence, modeling can be seen as a cyclic process (Overveld et al., 2015; Perrenet & Zwaneveld, 2012).

3.2.2 Simulation Modeling

Simulation modeling can be seen as a special case of modeling in which the model consists of a computer program and therefore is executable. In comparison to the mathematical modeling process, the simulation modeling process shows a computational — rather than mathematical — interpretation of the conceptualization, formalization and execution stages:

1. Conceptualization stage: Data are collected and a conceptual model is constructed and validated. In the process of abstraction, it is

decided what details to highlight and what details to ignore. Problem is formulated in a way that enables us to use a computer and other tools to help solve them (CSTA Computational Thinking Task Force, 2011).

2. Formalization stage: a computer program is constructed, i.e. requirements and specifications are stated and the system is implemented and tested (Comer et al., 1989). This includes making pilot runs, verifying the program and checking validity of the simulation model. If necessary the program is adjusted (Law, 2015). Thus, the formalization stage is a cyclic process in itself.
3. Execution stage: the model is being used for its purpose: designing and running experiments (Law, 2015).

Simulation modeling encompasses three methods: (1) System dynamics, associated with high level of abstraction where the individual objects are aggregated. The models can be described in terms of differential equations that are often non-trivial to solve. (2) Discrete event modeling, where the system modeled is considered to be a process, "i.e. a sequence of operations being performed across entities". The level of abstraction is lower. (3) Agent based modeling (ABM), which is made possible with recent growth of availability of CPU power and memory, does not assume any particular abstraction level. Agents have their properties and behavior and one can start building a model by identifying agents and describing their behavior even without knowing how a system behaves as a whole. ABM makes it possible to model systems that are difficult to capture with older modeling approaches (Borshchev, 2013) and it does not require familiarity with differential equations or mathematics beyond reach of secondary students. In our view, the characteristics of the ABM make it a suitable modeling method for our students who often lack deep understanding of the phenomena they model and make models specifically to deepen their understanding. To conclude, we consider conceptual representation which could be realized through the employment of ABM methods and software, in which "you give computational rules to individual agents and then observe, explore analyze the resultant aggregate patterns" (Wilensky, 2014) suitable for use in a secondary CS class "because the individual-level behavior of agents is relatively simple, [and] ABMs feature relatively simple computer programs that control the behaviors of their computational agents" (Wilensky, 2014).

In simulation modeling, repeating the conceptualization stage or going back and forth between the conceptualization stage and formalization stage are considered to be an integral part of the modeling process (Law, 2015). In the specific case of ABM, the boundaries between all modeling stages are blurred and it is considered a good modeling practice to develop a model in minute increments, cycling continuously through all modeling stages (Wilensky & Rand, 2015).

3.3 Method

The data were collected by the first author during the project-based lesson series. In view of existing studies involving algorithmic thinking and programming (see the introduction), we decided to use a combination of several data sources as a promising approach for our exploration of the students' activities and learning difficulties in their projects.

During their work in the class and the final presentations, screen and voice recordings were made of students' groups. (No recordings were made of students working elsewhere, such as at home). Except for a few corrupted recordings, they were all transcribed verbatim. The project documentation of each group was collected. After receiving their grades, twelve students filled in an online survey individually where they were asked about how they approached the work on this project, difficulties they encountered, what they have learned, what they liked or disliked, and what suggestions they had for the improvement of the assignment. Students were also invited to be interviewed. Five semi-structured interviews were conducted with individual students. The students were requested to describe their projects and they were asked if they could design a new NetLogo model on the fly (i.e., draw a sketch of the interface on paper and describe the model in terms of agents and interactions). Finally, they were asked what they learned during their work on the projects. The interviews were recorded and transcribed verbatim.

Using atlas.ti CAQDAS software we performed a qualitative analysis of the recordings, project documentation, survey results, and interviews, with coding categories based on the elements of our operational definition: *purpose, research, abstraction, formulation, requirements, specification, implementation, verification, validation, experiment, analysis,* and finally, *reflection*. After coding, we ascertained the visibility of the modeling elements in each of the data sources (see table 3) and examined the students' activities more in-depth, looking specifically for

indications of students' difficulties connected to each of the elements (see the following section).

3.4 Results

There were seven project teams. Five teams consisted of two students; one of three students, and one student opted to work by himself. Six of seven projects were successful; Team 5 did not finish theirs and did not turn in all the required project documentation.

Team 1 modeled chemical reactions and explored the resulting pH factor. Team 2 modeled lottery and explored how long people are willing to keep playing, depending on their winnings. Team 3 modeled the propagation of the Ebola virus and explored the possible effect of a vaccine on its propagation and the survival rate of those infected. Team 4 explored the influence of various factors to the length of time people stay at a party. Team 5 — who did not finish their project — set out to explore the influence of various factors on ice cream sales. Team 6 explored the growth and evolution of a colony of bacteria. Finally, team 7 explored whether mousetraps were more effective than cats in catching mice.

3.4.3 Results

We first present an overview of visible occurrences of the elements of our modeling operationalization, organized by data source and student (team): see table 3. Some elements were combined — see the descriptions below.

We now summarize the findings of our more in-depth analysis, organized by the elements of our operational description. We state our findings in general terms and illustrate them with characteristic text segments taken from the data.

Purpose. In the project documentation all teams clearly stated the purpose of their models. However, in the recordings we saw students tinkering with NetLogo and looking at existing models before deciding what phenomenon they wanted to model and explore. In answering the survey question whether it was difficult to decide what phenomenon to model and explore, four students answered affirmative and told us they had difficulties figuring out what could or could not be modeled. For S4a, who explored the behavior of partygoers together with S4b, the most important lesson learned during his work on this project was that it was important to have a clear idea of the purpose of the model before engaging in the modeling process — a thought shared by three other students in the survey.

		Purpose	Research	Abstraction	Formulating	Requirements and specification	Implementation	Verification and Validation	Experiment	Analysis	Reflection
Project documentation	Team 1	✓	✓	✓	✓	✓		✓			✓
	Team 2	✓		✓	✓	✓		✓			✓
	Team 3	✓	✓	✓	✓	✓	✓	✓		✓	✓
	Team 4	✓		✓	✓	✓		✓	✓		✓
	Team 5	✓		✓							
	Team 6	✓		✓	✓	✓		✓	✓		✓
	Team 7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Surveys	S1a		✓		✓		✓		✓		✓
	S1b								✓		✓
	S1c	✓	✓			✓	✓		✓		
	S2	✓	✓	✓	✓		✓	✓	✓		✓
	S3a	✓	✓	✓	✓		✓			✓	✓
	S3b	✓	✓	✓			✓	✓	✓		✓
	S4a	✓						✓			✓
	S4b	✓	✓	✓	✓		✓	✓	✓		
	S5a	✓	✓				✓		✓		✓
	S6a		✓								
	S7a		✓	✓		✓	✓		✓		✓
	S7b		✓	✓					✓		
Interviews	S1a	✓		✓	✓	✓	✓	✓	✓		✓
	S2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	S3a	✓	✓	✓		✓		✓		✓	✓
	S3b	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	S7a	✓		✓	✓	✓	✓	✓	✓	✓	✓
Recordings	Team 1	✓	✓		✓	✓	✓		✓		
	Team 2	✓		✓	✓		✓				
	Team 3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Team 4	✓	✓	✓		✓	✓	✓	✓	✓	✓
	Team 5	✓		✓	✓	✓	✓	✓	✓	✓	✓
	Team 6	✓	✓	✓		✓		✓	✓	✓	
	Team 7	✓	✓	✓	✓		✓				
	Presentations	✓		✓		✓	✓	✓	✓	✓	✓

Table 3: Frequencies of simulation modeling elements per data source per team or student. For example, Team 3 consists of students S3a and S3b.

Research. In the recordings we saw three students from two groups searching the Internet to learn about the phenomena they modeled. Team 3 reported in the documentation of their project about the possibilities to control the spread of the Ebola virus: *"Virus: does not spread through the air but through contact with an Ebola patient (sex, blood), slaughtering and eating of a sick animal, non-sterile*

needles. […] incubation about 21 days, 9 out of 10 people die", without reporting the source. In the survey, S3b mentioned consulting her sister who studied medicine. Team 6, exploring the effect of ambient warmth and the presence of food on life of bacteria, did not report any research in their project documentation. Others did not visibly engage in research but developed their models based on what they already knew about the phenomena they modeled (e.g. Team 1 who explored chemical reactions — in the survey S1a wrote they learned that in chemistry lessons) or their presumptions (e.g. Team 7, who explored whether mousetraps were more effective than cats in catching mice, or Team 5, who explored the influence of weather on ice cream sales).

Abstraction. All students engaged in abstracting: choosing a level of abstraction, based on the decision they made with respect to relevancy of particular features and deciding what to include into their models and what to leave out.

In the recording we observed several students struggling to determine such a level of abstraction. For example, Team 1 — who initially neglected the teacher's instruction to study the textbook first — had difficulties understanding the idea behind ABM and got 'stuck' in the notion of an aggregate state, e.g. thinking about pH as a contributing factor in a chemical reaction rather than the result of it. During the interview, S7a told us that he wanted his mice to reproduce but did not include this feature because he did not know how to implement males and females. It did not occur to him that gender of the mice was not relevant in his model. Finally, as required, all students who finished the project turned in wish lists with features or aspects that were not implemented yet but should be considered for the next version of the model, thus demonstrating they were able to decide what to include or leave out.

Formulation. The assignment required a description of the behavior of the model in a natural language, and all the students who finished their projects did that. However, several students needed help to formulate their problems appropriately: e.g., only after choosing the right level of abstraction did Team 1 manage to formulate their problem appropriately and, in the recordings, we heard S1a say, "*Two of these things have to collide with each other and then something needs to happen*".

Requirements and Specification. In the recordings it turned out to be hard to observe a distinction between requirements and specifications — see the results on Verification and Validation for a comprehensive example.

A description of requirements and specifications was a part of project documentation and all the students who finished their projects provided it. Team 7 wrote, *"The mousetraps need to be placed at random locations since we don't know what the perfect locations would be. If a mouse contacts a mousetrap, then the mouse needs to die/disappear."* In their project documentation, Team 1 stated requirements: *"In our program, two particles react to form two other particles. The probability that two particles react can be specified, as well as the reaction speed of the particles."* Then they wrote specifications extensively: *"If two initial turtles (red and yellow) meet, then the current catalyst value (the left slider) determines whether they react."* Team 3, who explored the propagation of the Ebola virus, wrote, *"In our model there is only one [breed of] turtles and it stands for people. These turtles can have various properties, such as being ill or healthy. They can be influenced by external factors such as medicine and their life span."*

While all students managed to implement something, some of them experienced difficulties. In the recordings we heard S1a say, *"I know what I want to do, but I don't know how to code it. I don't think it's all that difficult, but..."* During the interview, S7a told us he refrained from including mouse reproduction because he did not know how to design this feature and program it. When constructing their programs, only one team worked top-down: the others rather engaged in bottom-up incremental development constantly adding new features to their models.

Verification and Validation. The recordings revealed a complex picture in which the distinction between validation and verification was not always clear. Team 4's approach is representative of students' strategy: they constructed their model (program) by cycling among stating requirements and specifications, implementing and testing, in minute steps: *"We have to do that with time, man, that they can only drink one beer in ten seconds or so, otherwise they drink too much!"* When testing, it was not clear whether they were validating their model or verifying their code: often they would run their program, see remarkable behavior and subsequently change the code. S4b: *"All dead."* S4a: *"It begins to deteriorate now [in the simulation, the beer is gone and people leave the party quickly]"* S4b: *"But how could they all get the same amount of beer?"* S4a: *"That's because of that piece of code."* S4b: *"Really? Can't that be changed? How did they do it with the sheep*? *[Referring to an example from NetLogo's models library]"* Subsequently they would change their code and continue their work in a similar fashion. Team 6 worked similarly. It was not clear whether S6b was validating or experimenting: *"It works now but it is not balanced, so to speak."* S6a: *"Yeah."* S6b concluded: *"Yeah.*

That remains to be done" and went on to change the code. Later on they tried again. S6b: *"And if we make this one a bit lower, say seven or so, then they die, that is really abrupt, like, either they live or so, or all dead."*

In the project documentation, all of the students reported that their models behaved as expected (validation). Several students described validating their models and adjusting when necessary. To this end, Team 1 wrote: *"to prevent particles from reacting with each other immediately following a reaction, we built in a reaction pause. […] That way you prevent particles from being stuck in a constant back-and-forth reaction."*

Experiment. Team 7 was the only team who documented systematically performed experiments with their model: they reported the initial parameter values (e.g. ten cats and nine mice) and included the resulting data plots in their project documentation. In the recordings we saw other students engage in experimenting to various degrees, but most failed to mention this in the project documentation.

Analysis. Not all the students provided an analysis of the results of the experiments, but in the project documentation, they all reported answers to the purpose of their models. Team 7's analysis revealed, *"The mousetraps were not always effective. Some mousetraps go off but the mouse manages to escape."* Finally, they concluded that mousetraps were more effective than cats in catching mice. In the recordings we saw Team 3 analyze their data, without reporting it in the project documentation, and their conclusion was, *"We expected that the new medicine would decrease the spreading of Ebola. It turned out that the medicine worked rather quickly, but that the rate of infectiousness was of influence as well."*

Reflection. As required, all the students reflected on their models in the project documentation. Team 7 wrote: *"Not everything in our model corresponds with the reality. But it is nice to experiment with it. You can make your model as large and complex as you wish."* In the survey the students were asked what they learned. S3b wrote: *"It [modeling] is a good means to predict/research hypotheses. A good aid for research. I take chemical reactions as an example. You can make it and thus see (visualize) what happens,"* a thought reflected by S1a too. S4a learned that it was important to have a clear idea of the purpose of the model before engaging in the modeling process and that models and simulations never completely correspond to the reality. Contrary to S4a's reply, during the interview S1a expressed his astonishment about how easy it was to make a model that *"actually reasonably corresponded"* to what was modeled. He even went on to show it to his chemistry teacher.

3.5 Conclusion and Discussion

As to the **first research question** — *How can the intended learning outcomes of Computational Science (modeling and simulation) be described in operational terms* — we have obtained an operational description based on literature on modeling and simulation. The elements of the description turned out to be suitable to classify simulation modeling activities of the students in our study. However, some of these had to be grouped together since the separate elements could not be distinguished. The resulting operationalization contains the elements *purpose*, *research*, *abstraction*, *formulation*, *requirements/specification*, *implementation, verification/validation, experiment*, *analysis*, *reflection*. This 'blurring' of activities is also described by Wilensky and Rand (2015).

In answering our **second research question** — *What data sources are suitable to monitor students' learning outcomes when engaging in modeling activities* — we found that every source enabled us to observe some aspects of the modeling process. The interviews provided the opportunity to observe all the aspect of the modeling process, closely followed by recordings of students at work. In the project documentation, the description of the model and the reflection are well represented and experimenting and analysis not so: contrary to the presentations, where exactly the opposite happens. The surveys, in their present form, did not provide much insight into the modeling stages the students engage in.

We are planning to use our results to develop an assessment instrument. In order for such an instrument to be feasible for classroom usage, a combination of project documentation and class presentation are promising data sources that enabled us to capture all aspects of students' work. Our findings suggest that the instructions for documentation and presentation could be sharpened to improve visibility of (systematic) experimentation and data analysis within the model.

Finally, in answering our **third research question** — *What specific challenges do the students experience when engaging in modeling activities* — we identified some difficulties. Many students could not decide what to model exactly, and found it hard to decide on the level of abstraction and formulate the problem suitably for modeling through ABM. While all students managed to program something, not all of them were able to program all they wanted because either they could not decide on the relevance of a feature, or they did not know enough NetLogo to

code it. During testing it appeared to be difficult to attribute unexpected behavior to a fundamental modeling mistake, a programming error, or unexpected (i.e. *emergent*) behavior that was characteristic for the phenomenon under scrutiny. Students tend to rely on an incremental trial-and-error strategy while implementing their simulation model. Only a few conducted systematic and well documented experiments. Most of these experiments, together with the analysis of the results, were intermingled with the construction of the models.

This incremental development is consistent with description of the modeling practice, for example by Wilensky and Rand (2015). An ad hoc incremental development (trial-and-error strategy) is typical for novices (Robins et al., 2003).

General remarks. Although this was a small study with a limited number of participants, we learned a lot about students' understanding of modeling and simulation. Also, our findings indirectly informed us about the quality of the instruction, which leaves room for improvement. The instructional materials used were written with Artificial Intelligence (AI) and the role modeling could play in AI in mind. We feel they lacked specific focus and depth needed to teach modeling to a satisfactory degree.

Several students told us that through work on this project, they learned about the phenomena they modeled, which is in line with earlier findings (Blikstein & Wilensky, 2009; Taub et al., 2014). We often heard them laugh during their work and we observed that many students enjoyed working on this project. We saw that these CS students were able to utilize their CS/CT knowledge and skills to advance their learning in other disciplines.

In the subsequent phases of this research project, we will use these findings to explore CS teachers' initial pedagogical content knowledge of Computational Science in our second study (see chapter 4), and to develop instructional materials which will be used for the study on the assessment instrument (third study, see chapter 5) and the study on students' understanding and difficulties while working on Computational Science assignments (fourth study, see chapter 6).

Furthermore, we believe that the results of this research will contribute to the development of the CS curriculum in secondary education in the Netherlands, CS teacher training and CS education in general.

Chapter 4

Investigating Computer Science Teachers' Initial Pedagogical Content Knowledge on Modeling and Simulation

In this chapter, we describe our second study focusing on the CS teachers' pedagogical content knowledge (PCK) on modeling and simulation. We interviewed ten CS teachers and analyzed their PCK, distinguishing its four components — knowledge of (M1) goals and objectives, (M2) students' understanding, (M3) instructional strategies and (M4) assessment — and investigated potential differential features of their PCK in order to typify teachers' individual PCK. We charted the teachers' PCK in terms of these four components and found differential features related to knowledge of goals and objectives and related to knowledge of assessment, dividing these teachers into four distinct groups. However, these differential features do not lead to distinct types of PCK, thus not providing a typification that would allow to match each teacher's PCK to a distinct type of PCK. Our findings will be used to explore the future development of teachers' PCK and they will contribute to the development of teaching materials, assessment instruments and teacher training courses on modeling.

This chapter is based on the paper Grgurina, N., Barendsen, E., Suhre, C., van Veen, K., & Zwaneveld, B. (2017). Investigating informatics teachers' initial pedagogical content knowledge on modeling and simulation. In International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (pp. 65-76). Springer, Cham.

4.1 Introduction

Modeling plays a significant role in the development and learning of science (Justi & Gilbert, 2002) and CS provides the means for students to actively engage in learning science by providing tools and techniques to engage in modeling. The new 2019 Dutch secondary education CS curriculum recognizes this and includes an elective theme comprised of modeling and simulation, together called *Computational Science*. It is described by the high-level learning objectives: *"Modeling: The candidate is able to model aspects of a different scientific discipline in computational terms"* and *"Simulation: The candidate is able to construct models and simulations, and use these for the research of phenomena in that other science field."* Modeling itself will be a part of the compulsory core curriculum, described as *"Modeling: The candidate is able to use context to analyze a relevant problem, limit this to a manageable problem, translate this into a model, generate and interpret model results, and test and assess the model. The candidate is able to use consistent reasoning."* (Barendsen & Tolboom, 2016). The curriculum does not provide further details about these objectives, instruction or assessment. In line with the Dutch tradition, this is left to educators and authors of teaching materials. The elaboration of this learning objective, the development of teaching materials, assessment tools and teacher training courses are already taking place and the studies described in this thesis are an integral part of that effort.

Following Magnusson et al. (1999), we distinguish four components of content-specific pedagogy: (M1) goals and objectives, (M2) students' understanding and difficulties, (M3) instructional strategies, and (M4) assessment.

In our first study, we obtained an operational description of the intended learning outcomes of the learning objective Computational Science — thus focusing on Magnusson's component M1 about the goals and objectives, observed students working on modeling tasks — focusing on Magnusson's component M2 about students' understanding, and established what data sources were suitable for assessment — Magnusson's component M4 about methods of assessment (see chapter 3).

4.1.1 Aim of this Study

In this study, we turn our attention to teachers and focus on teachers' PCK on modeling. We address the following research questions:

1. How can the teachers' PCK be portrayed in terms of the four components of PCK?

2. What differential features of PCK can be used to identify patterns of individual PCK in terms of the four components of PCK?

The answer to the first question will serve as input to the second question that seeks to determine whether it is possible to recognize distinct types of PCK and subsequently match each teacher's PCK to one of these types.

Our findings will serve as input for the subsequent studies on designing teaching materials and assessment instruments, the development of teachers' PCK, and they will contribute to the development of teacher training courses.

4.1.2 Related Work

The construct of PCK has proven to be a powerful one to help capture teachers' views and knowledge on teaching various topics — both in STEM and in other disciplines — such as models and modeling in science (Justi & Gilbert, 2002) and as a part of public understanding of science (Henze et al., 2007), programing in CS (M. Saeli, 2012) and designing digital artifacts in CS (Rahimi et al., 2016), to expose the relation between the quality of teachers' PCK and their subject matter knowledge (M. Saeli, 2012; Sanders et al., 1993) to explore the PCK repertoire of beginning teachers (E. Lee et al., 2007) and to chart the development of teacher's PCK as the experience with teaching a particular topics increases (Henze et al., 2008).

We investigate PCK through the lens of the operational description of the intended learning outcomes of the learning objective Computational Science obtained in our previous study, describing the modeling cycle for simulation modeling through its elements *purpose*, *research*, *abstraction*, *formulation*, *requirements/specification*, *implementation, verification/validation, experiment*, *analysis*, *and reflection* obtained in our first study (see chapter 3.4).

4.2 Method

We conducted individual semi-structured interviews with ten CS teachers from the local CS teachers' network who replied to our invitation to be interviewed. Four of the teachers have an CS background and three of these are qualified teachers with an MSc degree in CS education while the fourth one is still studying to get this qualification. Out of the other six teachers, one has no teacher qualification, one is qualified for mathematics only, and other four are qualified

teachers for other subjects — such as physics or history — who additionally attained teacher qualification for CS through in-service training CODI scheme, described in chapter 2. The teaching experience of these ten teachers ranges from a few months to several decades. Eight of these teachers were about to take a course on agent-based modeling. Interviews lasted between half an hour and an hour, depending on the extent of a particular teacher's PCK. In our interview protocol, we first enquired about the teachers' educational and professional background and whether they had already taught modeling. We then asked detailed question arranged around the four PCK components described by Magnusson, that were inspired by Justi and Gilbert (2002, 2003), Rahimi (2016) and Henze et al. (2007).

- M1: Knowledge of goals and objectives

What comes to mind when you hear the word 'model'? In which context(s) does this word make sense to you? (Justi & Gilbert, 2003) What are models for? In which circumstances are they used? (Justi & Gilbert, 2003) What do you expect to be your main objective in teaching modeling and simulation in CS? (Henze et al., 2007) Why do you intend to teach this to your students in CS? What do expect you will like or dislike about modeling projects by your students? (Rahimi et al., 2016)

- M2: Knowledge of students' understanding

Do your students need any specific prerequisite knowledge to be able to learn about modeling and simulation in CS? (Henze et al., 2007) What sorts of skills do students need to acquire in order to be able to develop models and run simulations? (Rahimi et al., 2016) What do you expect to be successful for your students? (Henze et al., 2007; Rahimi et al., 2016) What do you expect to be difficulties for your students? (Henze et al., 2007; Rahimi et al., 2016) What do expect your students to actually learn from their modeling (and simulation) projects? (Rahimi et al., 2016)

- M3: Knowledge of instructional strategies

In what activities and in what sequence do you expect your students to participate in the activities of learning modeling and simulation? What to teach students to achieve the modeling objectives? (Rahimi et al., 2016) How to teach students to achieve the modeling objectives? (Rahimi et al., 2016) What do you expect to be your role as a teacher when teaching about modeling and simulation? (Henze et al., 2007) What do you expect are going to be the teaching difficulties/ problems concerned with the modeling projects in your classroom? (Rahimi et al., 2016) What technological tools do you intend to use in your classroom? (Rahimi et al., 2016)

- M4: Knowledge of assessment

How do you intend to assess your students' learning and achievement during their modeling projects? (Rahimi et al., 2016) How do you intend to establish whether your students reached the learning goals with regard to modeling and simulation? How would you know? (Henze et al., 2007)

The interviews were recorded and transcribed verbatim. We first coded the transcripts using the coding categories derived from our operational description of modeling: *purpose*, *research*, *abstraction*, *formulation*, *requirements/specification*, *implementation, verification/validation, experiment*, *analysis*, *and reflection* obtained in our first study (see chapter 3). We then classified the interview transcripts using Magnusson's four components of PCK (Magnusson et al., 1999) as main coding categories. Within these categories, we applied inductive coding to characterize the teachers' responses. In an axial coding process (Cohen et al., 2007), the codes were grouped and merged where necessary. We used the codes to describe the teachers' PCK in the results section (section 4.3). In a subsequent analysis, we tried to identify differential features in terms of Magnusson's components M1 through M4 in order to typify teachers' individual PCK.

4.3 Results

In this section, we first present the results of our characterization of the teachers' PCK organized around the four components of PCK (M1 through M4). Subsequently, we then explore differential features in order to distinguish types of teachers' PCK.

4.3.1 Knowledge about Goals and Objectives (M1)

No teacher has taught Computational Science as a separate topic in the context of the CS course yet and only one of them taught system dynamics modeling in a physics course. Since we did not enquire about the modeling process explicitly, we performed a detailed analysis of the interviews through the lens of our theoretical framework on the modeling cycle and thus reconstructed the teachers' content knowledge (CK) pertaining to the aspects of Computational Science obtained in our first study (see chapter 3), as shown in Table 4.

Aspect \ Teacher	1	2	3	4	5	6	7	8	9	10
purpose		x		x			x	x	x	x
research									x	
abstraction	x								x	x
formulating	x			x			x	x	x	x
requirements/ specification	x			x						
implementation				x		x	x		x	
verification/ validation	x	x	x		x	x		x		x
experiment	x	x		x					x	x
analysis				x			x	x	x	
reflection	x	x		x		x	x		x	x

Table 4: Teachers' PCK on modeling cycle

Concerning the nature of models, six teachers said that models were a simplified representation of reality, three reported that models were something to be used for calculations, two saw models as something for visualization and communication, and one also mentioned physical models. Regarding the contexts for the use of models, the teachers mentioned information systems, scientific research and cited several examples (e.g. exploring group forming on ethnical basis). Several teachers stressed the fact that CS served other disciplines — an idea that persistently permeates their thinking about modeling and teaching modeling.

According to the teachers, the objectives of teaching modeling are twofold:

- **Conceptual objectives**: these objectives emphasize learning to master skills associated with CS subject matter. Teachers mention CT aspect automation, software design cycle, the necessary research skills, analysis of the world the students live in, linking and translating reality to model through abstraction, building a model and, in words of teacher 9, *"using it […] so that you can predict things or test things, or … ehm …, understand things"*.
- **Motivational and practical objectives**: these objectives focus on students' engagement, motivation, attitude, skills, practical benefits, insight and awareness about relevance. Teachers mention perseverance, building confidence about students' own ability, developing self-reliance, and realizing that models are useful tools that can be used in specific situations. In this light, learning to model helps to enhance students' insight, it helps develop cooperation and communication skills, it is interesting or fun, it serves as preparation

for the future, and it concerns "soft CS". Furthermore, modeling lays a connection between CS and other disciplines and models could be useful for students' subsequent studies. In words of teacher 10, *"… that you need to be able to design a test setup with CS, ehm, digital means so that you can get a lot of results, but processing these results is no more CS, that's physics."*

4.3.2 Knowledge of students' understanding (M2)

The prerequisite knowledge needed to learn modeling and skills needed to make models are related to:

- **Student's characteristics**: such as age and development, mindset, attitude, and other skills. For example, teacher 6 believes *"it is all very complex, they [students] are only 15 years old"* and teacher 7 is reminded of Bloom's taxonomy and fills in *"it gets easier as they get older"*. Fantasy, creativity and an analytical mindset also play a role. Attitude is important: several teachers stress the significance of perseverance. Some mention differences among students: teacher 6 expects to see different skill levels when it comes to tackling practical assignments.
- **Student's knowledge pertaining**: the other discipline, programming or computational thinking, and modeling aspects. Teachers 4, 5, 6 and 8 expect the students to know something about the phenomenon from other discipline they are modeling. Almost all the teachers expect their students to be familiar with programming before they embark on modeling. However, teacher 9 also sees the possibility to use modeling as a vehicle to teach programming. Teachers 4, 8 and 10 expect their students to be familiar with several modeling aspect such as being able to explain how a particular model works, abstraction, and in words of teacher 8, *"you need to learn to recognize the actors and you need to be able to see the relations among them, to …ehm … to describe them, to translate them into a model"*.
- **The chosen teaching approach**: teacher 8 mentions the interplay between the teaching materials used and the necessary prerequisite knowledge and says *"but you need to sense what extras you should offer"* and adds, *"It's quite abstract so I think about the 12th grade, but if you present it simply enough, then you could teach it in the 10th*

grade as well, without ever having taught any CS before that". He also stresses the importance of re-activating the knowledge the students already possess.

The issues teachers regard as successful:

- **Relevance:** through modeling, learning about familiar phenomena encountered in other courses (e.g. biology, economy) rather than distant ideas. As teacher 6 puts it: *"You concoct nice stories about nuclear power plants, but how many children end up in there?"*
- **Perception:** attitude, experiencing success, interest and fun, and student's characteristics. Several teachers expect students' confidence would grow through their perseverance when facing problems and experiencing success by making a model on their own. Teacher 2 admits that achieving this with his students poses a challenge. Numerous teachers expect these aspects, together with the relevance experienced by students and the possibility to come up with creative solutions and implementation of students' own ideas, to make modeling interesting and fun. Teacher 10 mentions Gardner's multiple intelligences to explain why he expects students in science tracks to perform differently than students in humanities tracks.
- **Skills:** programing and computational thinking. Most teachers expect that programming the models would not pose a problem.
- **Organizational issues:** teaching strategies to meet students' needs. Several teachers describe how to contribute to the students' success by choosing a suitable teaching approach. A number of teachers who intend to use NetLogo to teach modeling prize its user-friendliness and see this as a success factor.
- **Interest and fun:** Teacher 5 says, *"even if I never teach this, I still find it fun for myself"* and goes on to reflect how to transfer this enthusiasm to his students.
- Finally, teachers 1, 3, 4 and 9 do not know what to say. As teacher 4 puts it, *"I've never taught modeling, so I wouldn't know what would be a success.*"

The issues teachers regard as difficult:

- **Technical issues:** programing and computational thinking, aspects of modeling and development. For example, the meaning of the

term *parameter* in physics or mathematics differs from the meaning in a simulation model and that might be confusing. Students who follow physics course are already familiar with models, as opposed to students in humanities tracks. Teacher 2 expects that in the beginning, students will have difficulties understanding existing models and how their components interact. Teacher 4 expects her students to have problems getting used to the programming language and subsequently to have problems programming. Numerous teachers expect that modeling aspects such as abstraction – deciding what is relevant for a model and what to leave out — will be difficult. Some teachers expect problems with implementation — translating conceptual model into program code. Teacher 7 expects his students could be too ambitious with the models they want to make and suggests keeping an eye on his students all the time to be able to intervene and help in case they encounter any of these problems.

- **Perception:** attitude, skills, interest or fun, relevancy, age and development. Teacher 9 expects problems with motivation if the students do not see the relevance of modeling. He also mentions lack of perseverance and inability to go on after getting stuck. Teacher 6 believes that abstract aspects of modeling are difficult for the students of this age – 16 years old. He also mentions students not using common sense.
- **Approach**: work method, possibility to work on their own case. Some teachers expect problems with students who dive right into building their models without giving it sufficient thought first, and, with students who lack oversight and do not know where to begin. According to teacher 9, the last problem could be alleviated by good teaching material. Several teachers believe some students would have difficulties coming up with a suitable case to model.

Again, some teachers do not know what to say.

4.3.3 Knowledge about instructional strategies (M3)

When asked about their teaching approach, some teachers are cautious about replaying due to lack of experience, but in the end all the teachers have similar ideas that can be summarized as follows: during a period ranging from six to twelve weeks, scaffold learning by beginning with an introduction about models

in general, show and explain the working of few models and their code, then give students several assignments to expand existing models or develop simple models from scratch, possibly differentiate to account for students interests, level (HAVO or VWO) or the track they follow (humanities or science), and finally, engage them in a large (group) project where they develop a model from scratch during several weeks. While describing their teaching approach with various degree of detail, the teachers report about:

- **Their role as a teacher:** to coach, to explain, to show, to help, to encourage, to keep an eye on students' progress, and simply to be there. Teacher 9 says, *"and then you come in there to steer. And if it gets stuck, not just to answer the question, but to help them find the answer themselves"*. Teacher 5 is cautious: *"well, when they're working on a new model, I don't always have a ready-made answer."* Teacher 7 would have his students recreate a small program to check their understanding before they embark on the large project. During the project, he would have a double role, both as the teacher and as a client. As the teacher, he would keep an eye on the progress of the whole project. As the client, he would come in every two weeks and tell his students things like *"hey, you have things that make no sense"* and have students fix the problems themselves.
- **Assignments the students work on:** open and closed problems and making models. Teacher 6 would have a number of closed problems together with answers for students to practice before embarking on the open problem they have to work on as their final project. For the final project, most teachers would let their students come up with their problems themselves, but would also have a list of problems available for those who cannot think of something themselves.
- **Student's characteristic to take into account:** level and the track they follow. Teacher 8 would have different assignments for students to practice on, catering to their needs and preferences, depending on the educational track they follow. Teacher 2 would not require his HAVO students to develop a model from scratch, but would rather have them expand and adjust an existing model.
- **Organizational aspects:** the daily teaching practice, planning, organizing, SCRUM[20], rapid prototyping and playing the role of the

20 https://www.scrum.org

client. Teacher 3 teaches 60-minutes lessons and would start each lesson with a short central instruction and let the students work for themselves the rest of the time. He would use Trello boards for planning and, like teacher 9, employ SCRUM to organize the work. Teacher 10 would send his students to a teacher of a different subject who would then pose as a client, while he himself would be a process supervisor. Teacher 9 would pose as a client himself.

- **Difficulties and problems:** technical problems, problems related to teaching materials and other problems. Teacher 9 warns that no matter how simple the software used is, there is always a possibility of getting an error message, not understanding it and then getting stuck.

4.3.4 Knowledge about Assessment (M4)

When asked about assessment, all teachers agree that the large practical assignment the students worked on to learn modeling could serve for the assessment purpose as well. Teacher 9 could possibly give his students a written exam instead. Teacher 8 would use a small written exam to ascertain the students learned enough about modeling before they are allowed to start working on a large project. When talking about assessment, the teachers report on:

- **Assessment form:** written exam — formative and summative, project to be done individually or in groups.
- **Problems to work on:** given by the teacher or provided by the students themselves, open or closed, opportunity for differentiation.
- **Organizational issues**: SCRUM and rapid development.
- **Assessing the work**: quality of the end product and project documentation, aspects of modeling cycle, teacher's impression about the students' activities in the lessons during work on the project. When assessing the results of the final project, most teachers want to look into the quality of the model — and some of them the code too — as described by the students in the project documentation they are required to turn in. The teachers mention various aspects of the modeling process as relevant for the assessment. However, no clear quality criteria are elaborated and teacher 1 would simply estimate the quality of the project. Teacher 8 adds that technical aspects are easy to assess, but it is difficult to see if the students realize the full

spectrum of possibilities the modeling offers to them. Some teachers would rely mainly on their observation of students while they work on the projects, again without elaborating on specific quality criteria. Additionally, teacher 10 would talk to his students to ascertain whether they understand what they are doing. Teacher 6 would also assess the quality of the report the students wrote for the customer. Teacher 7 would have his students present their models, he would assess the product (i.e. model and project documentation) and additionally the process (SCRUM) and he stresses the importance of reflection, together with number of other teachers. He adds, *"even if they didn't succeed, I find you can't say they don't know what modeling is"*. Teacher 5 is not sure what to say: "*if the model works, is that sufficient?"*

4.3.5 Differential Features and Typification

We found two characteristics that distinguish among teachers:

- **Knowledge of goals and objectives (M1).** Teachers 1, 2, 4, and 6 stress the importance of *conceptual objectives* such as learning how to employ programming or how to make models, while teachers 3, 5, 7, 8, 9 and 10 put more emphasis on *broader motivational and practical objectives* such as enhancing insight or preparing for the future.
- **Knowledge of assessment (M4).** Teachers 1, 2, 4, 8, 9 and 10 predominantly put to use *product-based assessment* stressing the quality of the students' product, while teachers 3, 5, 6 and 7 prefer *process-based assessment* stressing the importance of the employed working procedures.

Combining these two differential features leads to four groups of teachers as shown in Table 5:

		M1	
		Conceptual objectives	Practical and motivational objectives
M4	Product	1, 2, 4,	8, 9, 10
	Process	6	3, 5, 7

Table 5: Distinct groups of teachers

Our analysis revealed, however, that the knowledge of students' understanding (M2) and instructional strategies (M3) varies within each of these four groups, so the above differential features do not give rise to a typification of the teachers' overall PCK that would allow to match each teacher's individual PCK to a distinct type of PCK.

4.4 Conclusion

In answering the **first research question** — *How can the teachers' PCK be portrayed in terms of the four components of PCK?* — we portrayed each of these components:

Concerning teachers' **knowledge about goals of objectives** on teaching modeling, we charted teachers' content knowledge and described learning objectives in terms of conceptual, and motivational and practical objectives.

Concerning the **knowledge about students' understanding**, the teachers reported on the prerequisite knowledge, attitudes, skills, abilities and various approaches students have to learning, in line with Magnusson et al. (1999). However, when talking about the difficulties, they mentioned problems due to the abstract nature of modeling and inefficient students' strategies, but no teacher mentioned misconceptions.

Concerning the **knowledge about instructional strategies**, we see an agreement about subject-specific strategies (Magnusson et al., 1999) — scaffolding learning with a final project which serves both to give students the opportunity to learn how to develop a model from scratch and as assessment.

Concerning **knowledge of assessment**, there is an agreement about a suitable form — a large practical assignment. Teachers mentioned a range of assessment criteria focused on the quality of students' products and teachers' impressions of the students work process. In contrast to the uniformity regarding the form of assessment, there is a great variation in granularity and depth of their description of assessment criteria. We saw similar diversity regarding the knowledge of dimension to assess (Magnusson et al., 1999), i.e. the aspects of modeling process.

In answering the **second research question** — *What differential features of PCK can be used to identify patters of individual PCK in terms of the four components of PCK?* — we found two characteristics that distinguish among teachers: their

focus on conceptual versus motivational and practical objectives (M1) and their emphasis on product-based versus process-based assessment (M4) leading to four distinct groups of teachers. However, none of these differential features leads to an overall typification of the teachers' PCK.

4.5 Discussion

Reflections on findings. As a possible explanation for the variations found in the teachers' PCK, we explore the relation with the teachers' background. Not surprisingly, we saw that teachers 1, 2, 4 and 9, who all have a background in CS, displayed rich knowledge of modeling. Teachers 1 and 4 — both young teachers — had limited idea about what to expect from their students beyond the general remarks about students needing to plan before acting and lack of perseverance. On the other hand, teachers 2 and 9 exhibited rich knowledge of students' understanding and related their extensive knowledge of instructional strategies to it. Teachers 6, 7, 8, and 10 possess rich and well-connected PCK. Finally, we saw that teachers 3 and 5, despite their limited content knowledge, were able to relate their general knowledge of their students to their teaching strategies. The knowledge of their students' understanding and difficulties was the strongest component of their PCK. Other components of their PCK were weaker and so were the relations among these components too.

Regarding the instructional strategies, the teaching approach described here is in line with prevailing CS teaching practices in the Netherlands (Schmidt, 2007). The extent of knowledge of topic-specific strategies (Magnusson et al., 1999) varies across teachers and seems to be related to their subject matter knowledge and teaching experience. The findings about young teachers 1 and 4 are in line with the results by Lee et al. (2007) who found that *"a strong science background does not guarantee a proficient level of PCK."* The more experienced teachers sometimes behave like novices too, — e.g., teacher 5 — while in others, their extensive PCK seems to sustain them in the non-familiar area of modeling, in line with results of Sanders et al. (1993) who found in a similar situation that rich PCK for general science topic seems to sustain teachers *"in whatever content they are teaching"*.

We observed two characteristics allowing us to distinguish among teachers that were observed by Rahimi et al. (2016) as well — in the knowledge of goals and objectives (M1), with preference for either conceptual objectives or broader

motivational and practical objective, and in the knowledge of assessment (M4), with preference for either product-based assessment or process-based assessment. However, our findings are not completely in line with those findings because in our case, for example, teachers 1 and 10, both leaning toward predominantly product-based assessment, require students to keep logbooks to document the modeling problems, difficulties and dead ends they encountered. On the other hand, teacher 6 would take customer's feedback into account and teacher 7 would have his students present their work in the class, while they both lean toward predominantly process-based assessment. Unlike Rahimi et al., we were not able to typify teachers' PCK through relating their knowledge of students' understanding and instructional strategies on one hand, to their knowledge of goals and objectives and knowledge of assessment on the other.

Remarkably, despite the great variation of assessment criteria mentioned, there is hardly any evidence of quality indicators used to judge to what extent these criteria are met and to what extent the students are able to apply the elements of modeling to a satisfactory degree.

Limitations of the study. In this study, we charted PCK of a small group of CS teachers. However, because of the variations in their educational background, teacher qualification and teaching experience we expect that our findings are fairly typical for the population of Dutch CS teachers. This is to be confirmed in further research.

Implications for educational development. We believe that teachers would benefit not only from a course on modeling, but also from the availability of teaching materials. We are convinced that the quality of assessment — an issue attracting a lot of attention in modern CS education (Alturki, 2016) — would improve if teachers get assistance with designing assessment instruments that would take into account both product and process.

In the subsequent phases of this research project, we will use these findings when we focus on the development of teaching materials (the fourth study, see chapter 6) and assessment instrument (the third study, see chapter 5). In parallel, in-service teacher training based on these findings will be offered to interested teachers. Finally, all the participants in this study will be followed to chart the development of their PCK of Computational Science in the future — an endeavor that falls beyond the scope of this thesis.

Chapter 5

Assessment of Modeling and Simulation in Secondary Computing Science Education

Using the findings from the first two studies, we developed a curriculum intervention including a practical assignment and an accompanying assessment instrument consisting of grading rubrics based on the SOLO taxonomy. In this chapter we focus on the assessment instrument. We describe its development and report on a pilot study carried out in the secondary computing science course implementing the curriculum intervention. The instrument proved to be reliable and effective in tracing high and low levels of the students' achievements in modeling and simulation projects and exposed the expected differences in performance levels of various groups of students, which renders it useful for both formative and summative assessment. Furthermore, our application of the instrument has provided new insights into the needs of specific groups of students to receive instruction prior to and during the work on the assignments.

This chapter is based on the paper Grgurina, N., Barendsen, E., Suhre, C., Zwaneveld, B., & van Veen, K. (2018). Assessment of modeling and simulation in secondary computing science education. In Proceedings of the 13th Workshop in Primary and Secondary Computing Education (pp. 1-10) (Grgurina, Barendsen, Suhre, Zwaneveld, et al., 2018).

5.1 Introduction

The new 2019 secondary computer science curriculum recognizes the importance of modeling and includes an elective theme comprised of modeling and simulation, together called Computational Science. It is described by the high-level learning objectives: *"Modeling: The candidate is able to model aspects of another scientific discipline in computational terms"* and *"Simulation: The candidate is able to construct models and simulations, and use these for the research of phenomena in that other science field."* (Barendsen & Tolboom, 2016). The curriculum does not provide further details about these objectives, instruction or assessment. In line with the Dutch tradition, this is left to educators and authors of teaching materials. The elaboration of these learning objectives, the development of teaching materials, assessment tools and teacher training courses are already taking place. We participate in these endeavors with practical assistance to teachers developing teaching practices that help to attain these objectives, and by monitoring these developments through research described in this thesis. In our first study, we obtained an operational description of the intended learning outcomes of the learning objective Computational Science — thus focusing on Magnusson's component M1 about the goals and objectives, observed students working on modeling tasks — focusing on Magnusson's component M2 about students' understanding, and established what data sources were suitable for assessment — Magnusson's component M4 about methods of assessment (see chapter 3). In our second study (see chapter 4), we investigated teachers' initial pedagogical content knowledge on modeling and simulation. We then proposed an assessment instrument (Grgurina, Barendsen, Suhre, Veen, et al., 2018) pertaining to Magnusson's component M4 about methods of assessment.

In this study, we further focus on monitoring the levels of understanding in the learning outcomes of students engaging in modeling projects — Magnusson's component M4. We aim to examine the agreement and validity of that assessment instrument to assess students' proficiency in modeling a (problematic) situation and to provide answers to improve the situation. We seek answer to these questions:

1. Can the instrument be used by different teachers without having a distinguishable effect on the assessment?
2. Does the instrument allow for a valid measurement of students' proficiency level?

Two steps are taken to obtain the necessary data to answer both questions. First, inter-rater agreement is assessed by having two teachers rate the same products and compute the inter-rater agreement. Second, the proficiency levels of projects made by students of different education level — HAVO and VWO — are compared to investigate whether differences are in the expected direction.

5.2 Background and Related Work

5.2.1 Computational Thinking: Modeling

Formulating problems in a way that enables us to use a computer to solve them and representing data through abstractions such as models and simulations are integral parts of computational thinking (CT) (CSTA Computational Thinking Task Force, 2011). With the arrival of computers into schools, new venues are created to support students' learning in various disciplines through the use of computer models, i.e. models that are implemented and run as computer programs (Blikstein & Wilensky, 2009; Overveld et al., 2015). Wilensky argues, "Computational modeling has the potential to give students means of expressing and testing explanations of phenomena both in the natural and social worlds" (Wilensky, 2014), as do Caspersen and Nowack (2013b). Indeed, modeling plays a significant role in the development and learning of science (Justi & Gilbert, 2002) and CS equips the students to actively engage in learning science by providing tools and techniques to engage in modeling, thus enabling them to provide meaning to the learning both of the discipline at hand (Gilbert, 2006) and CS.

In the new 2019 CS curriculum, for the intended learning outcomes of the learning objective Computational Science, in one of our previous studies we developed an operational description that describes the modeling cycle for simulation modeling through its elements purpose, research, abstraction, formulation, requirements/specification, implementation, verification/validation, experiment, analysis, and reflection. Furthermore, in that study we advocated to use agent-based modeling (ABM) when teaching Computational Science since it is a suitable simulation modeling method for use in secondary CS education (Grgurina et al., 2016) and here we focus specifically on ABM type of computer models.

5.2.2 Documenting Models

In order to describe what is the purpose of a model, how does it work and other relevant details, it is necessary to document the model. Several techniques have been proposed to do this in order to help to understand a model, to facilitate completeness of the description, and to make it easier to reproduce a model.

The ODD protocol is specifically devised to standardize the descriptions of individual-based and agent-based models (Grimm et al., 2006, 2010). It describes a model in terms of its Overview, Design concepts and Details — hence the acronym ODD. In the updated ODD protocol, the overview contains the elements (1) purpose, (2) entities, state variables and scales, and (3) process overview and scheduling. The eleven elements of the design concepts are the basic principles, emergence, adaptation, objectives, learning, prediction, sensing, interaction, stochasticity, collectives and observation. Finally, the details deal with the elements initialization, input data and submodels. However, this approach to documenting models has several weaknesses: due its textual nature, it is inherently ambiguous and furthermore, it does not allow for documentation of all the relevant details and thus hampers the reproduction of models, as noted by Amouroux et.al (2010). They charted the strengths and weaknesses of the ODD protocol and suggest the addition of an *"Execution environment"* section to support the model replication. A different approach to documenting models recognizes the common traits of agent-based modeling and object-oriented programming and suggests to expand the Unified Modeling Language (UML) to accommodate the specifics of ABM. The UML supports the following kinds of models: static models, dynamic models, use cases, implementation models and object constant language (OCL) (Rumbaugh et al., 2004). Odell et al. (2000) suggest the agent-based extension AUML, that is, *"agent-based extensions to the following UML representations: packages, templates, sequence diagrams, collaboration diagrams, activity diagrams, and state charts."* Similarly, Bauer et al. (2001) propose Agent UML with four agent-based extensions to UML representations: packages, templates, sequence diagrams and class diagrams. Muller et al. (2014) go a step further to explore the suitability of particular types of model descriptions for specific intended purposes. To this end they distinguish eight possible purposes of models: communication of the model — to peers, for education or for stakeholders; in-depth model comprehension, model assessment — to establish its suitability for its purpose, model development — design and collaborative, model replication, model comparison, theory building, and finally, code generation. They go on to assess how well these purposes are met

by the different description types: natural language — either with a prescriptive structure, such as ODD protocol, or without it, such as verbal description; formal languages — ranging from various ontologies, source code, pseudo code to mathematical description, and finally graphics — either formal such as UML, or non-formal. In case of communication for education, they suggest that non-formal verbal description, source code made with the program-level tools (such as, for example, NetLogo (Wilensky, 1999)) and non-formal graphics are among the most suitable description types, while formal descriptions with ODD protocol or UML as well as non-specialized programming languages are less suitable. They conclude by suggesting "a minimum standard of model description".

5.2.3 Assessment

Brennan and Resnick focused on assessment of the development of CT during learning in informal settings and developed a CT framework distinguishing three dimensions: computational concepts describing the concepts designers employ when programming, namely *sequences, loops, parallelism, events, conditionals, operators, and data*; computational practices describing the practices designers employ when engaging with the concepts, namely *being incremental and iterative, testing and debugging, reusing and remixing, and abstracting and modularizing*, and computational perspectives describing the perspectives designers form about the world around them and about themselves, namely *expressing, connecting and questioning* (Brennan & Resnick, 2012). Zhong et al. (2016) brought these three dimensions of CT into the classroom when designing an assessment framework for elementary school students and they redefined them as follows: computational concepts as "*objects, instructions, sequences, loops, parallelism, events, conditionals, operators, and data*"; computational practices as "*planning and designing, abstracting and modeling, modularizing and reusing, iterative and optimizing, and testing and debugging*", and computational perspectives as "*creative and expressing, communicating and collaborating, and understanding and questioning*". Using this framework, Lye and Koh (2014) analyzed 27 intervention studies in K-12 aiming at the development of computational thinking through programming and found that the majority focuses on computational concepts and only six on computational practices. In order to promote focus on computational practices and computational perspectives in a K-12 classroom, they suggest an instructional approach providing "*a constructionism-based problem-solving learning environment, with information processing, scaffolding and reflection activities*." Brennan and Resnick

offer six suggestions for assessing computational thinking via programming, among others to make assessment useful to learners, to incorporate creating and examining artifacts, and to have the designer illuminate the whole process (Brennan & Resnick, 2012). These views are corroborated by the findings in our prior study on CS teachers' pedagogical content knowledge (PCK) of modeling and simulation, where we learned that the interviewed teachers mostly suggest a hands-on approach to learning and that the preferred assessment form for most of them would be a practical assignment lasting several weeks, where student groups would construct models and use them to run simulations and conduct research while extensively documenting the whole process. At the same time, we observed a great diversity in the assessment criteria teachers mentioned, but very few corresponding quality indicators used to judge to what extent these criteria are met (Grgurina et al., 2017).

In the eyes of the students, the assessment defines the actual curriculum, according to Biggs and Tang (2011). In their constructive alignment network, the curriculum is stated in the form of clear intended learning objectives (ILO) specifying the required level of understanding, the teaching methods engage students in doing things nominated by the ILO's, and the assessment tasks address these ILO's. Learning outcomes can be classified using the Structure of the Observed Learning Outcome (SOLO) which describes the learning progress through five levels of understanding. The first three levels — prestructural, unistructural and multistructural — are considered to be quantitative in the sense that prestructural indicates missing the point, unistructural means meeting only a part of the task and multistructural shows a further quantitative increase in what is grasped: *"knowing more"*. Relational, on the other hand, indicates a qualitative change indicating conceptual restructuring of the components — *"deepening understanding"*, and extended abstract takes the argument into a new dimension (Biggs & Tang, 2011).

Looking into the use of SOLO taxonomy to assess the novice programmers' solutions of code writing problems, Whalley et al. (2011) noted that previous research had indicated difficulties in mapping from student code to the SOLO taxonomy *"since the mapping process seems very context bound and question specific"*. Indeed, Meerbaum-Salant et al. (2013) remarked that while the strength of the SOLO taxonomy lies in the fact that it offers a holistic, rather than a local perspective, "using [it] for various types of activities, simultaneously, is not straightforward". When they set out to combine the Revised Bloom taxonomy

(Krathwohl, 2002) with SOLO in order to construct assessment for programming tasks of novice programmers, they started out with the interpretation of SOLO as five ordered categories:

- *Prestructural*: Mentioning or using unconnected and unorganized bits of information which make no sense.
- *Unistructural*: A local perspective — mainly one item or aspect is used or emphasized. Others are missed, and no significant connections are made.
- *Multistructural*: A multi-point perspective — several relevant items or aspects are used or acknowledged, but significant connections are missed and a whole picture is not yet formed.
- *Relational*: A holistic perspective — meta-connections are grasped. The significance of parts with respect to the whole is demonstrated and appreciated.
- *Extended abstract*: Generalization and transfer — the context is seen as one instance of a general case (Meerbaum-Salant et al., 2013)

The issue of assessing the learning of the students engaged in larger programming projects attracts attention as well. Casto and Fisler (2017) explored how to track program design skills through an entire CS1 course at university level and suggest a multi-strand SOLO taxonomy, thus corroborating the idea that using SOLO taxonomy simultaneously for various types of activities is not straightforward. They suggest a multi-strand SOLO-taxonomy without the extended abstract level since none of the students in their study reached that level (Castro & Fisler, 2017). A multi-strand SOLO taxonomy is in line with the idea that one assessment task might address several ILO's and vice versa, one ILO might be addressed by several assessment tasks (Biggs & Tang, 2011). Assignments for complex tasks encompassing diverse ILO's — such as in case of Computational Science, thus going through a modeling cycle by formulating a problem, pinpointing the research question, designing a model and using it to answer the research question — warrant the elaboration of criteria defining performance for each of the ILO's involved.

5.3 Assessment Instrument

Based on these findings, we developed constructionist teaching material about agent-based modeling with NetLogo meant for the CS students in the 11th and 12th grades of both HAVO and VWO who are preferably no novice programmers but rather somewhat experienced, probably in other programming languages. The teaching material covers all the aspects of the ILO's of Computational Science we identified earlier (Grgurina et al., 2014b), and addresses not only computational concepts such as programming to implement the model, but also computational practices such as the validation of the model and computational perspectives such as formulating the research question to be answered through the use of the model. Together with this teaching material, we also developed an assessment instrument on which we focus here.

Following the teachers' suggestions about the desirable form of assessment (Grgurina et al., 2017) and our findings about suitability of various data sources for assessment confirming the suitability of project documentation (Grgurina et al., 2016), we developed an assessment instrument consisting of a practical assignment where students design models and use them to conduct research of phenomena in another science field, and accompanying rubrics for assessment based upon the project documentation and models themselves (i.e. program code). Guided by the suggestions for the rubrics construction by Wolf and Stevens (2007), from the modeling cycle we first identified the criteria that defined performance as: stating the case and the research question, designing the model and implementing it, validation, experiment, analysis, answering the research question, and reflection. Subsequently, we designed a practical assignment that provides several cases and research questions for students to choose from, a detailed description of the modeling process they need to engage in, and a corresponding rubrics based on SOLO taxonomy.

An example of the cases provided is the question whether sustainable human life is possible on Mars. The students are pointed to the websites of NASA and SpaceX to learn about the current state of affairs and subsequently have to explore whether, after the initial supplies and shelter were delivered, it would be possible to produce sufficient water, air and food to survive and thus whether it would be possible to found a sustainable human colony on Mars. Among other cases are the questions, what is better for traffic flow on a junction: a roundabout or traffic lights, and to investigate the optimal number and task division of bank counters as

to minimize the waiting time of the customers with various needs. In line with our dedication to stimulate student engagement, the students are allowed to come up with their own research questions instead.

While these assignments allow students to make their own choices and decisions when designing their models, we needed a standard that allows educators using our assessment instrument to easily assess the quality of their students' models. To set such a standard, for each case we constructed a minimal expert model — a description of a minimal model that fulfills the stated purpose and contains all the necessary agents with their correct behavior and interactions. Since we wanted these minimal expert models to be described on a conceptual level only, we refrained from implementing them in NetLogo because we believed that that would hinder the assessment process rather than contribute to it. Instead, for the models that our students are expected to make — two-dimensional, containing only a few types of agents, no links and no advanced behavior such as learning or sensing — we devised our own format to describe them. This format is partly narrative, borrows aspects of class diagrams from UML and exploits the idea of graphical representation of timed automata where it is possible to require that particular state transitions are allowed only under certain conditions, or only synchronously with other state transitions (Vaandrager, 2011). Here we illustrate this approach with the description of our minimal expert model of the roundabout. First of all, there is the agent type[21] vehicle with its representation — inspired by UML class diagram — stating that an agent of this type has properties[22] current position and target position, and behavior consisting of actions show up, wait, move and leave.

Vehicle
current position target position
show up wait move leave

Figure 3: UML class diagram for *vehicle*

Then there is a graphical representation of a state diagram of a *vehicle* (figure 4) which is interpreted as follows: during the NetLogo *setup* procedure, the first

21 NetLogo speaks of breeds of agents, but we use the term type to cater for those not familiar with NetLogo.

22 NetLogo speaks of agent's own variables

transition from the state *begin* to the state *ready* occurs, and that is the action *show up.* The NetLogo *go* procedure then runs repeatedly until the program stops and according to our convention, every time it runs, exactly one transitions originating from the *ready* state takes place, and when the procedure has finished one run the *vehicle* is back in the *ready* state unless it reached the *gone* state. This is not to say that during one run of the *go* procedure an agent may engage in one state transition only: rather, it means that one full cycle of actions emanating from the *ready* state takes place. For each transition, there might be conditions that need to be met in order for the transition to occur, and properties could be updated. For example, *move* only occurs if the position in front of the *vehicle* is free and then its property *current position* is updated. State transitions could be synchronized with each other as well, such as in the famous Wolf Sheep Predation model: a *wolf* can *eat* only if one of the *sheep* simultaneously *dies* (Wilensky, 1997).

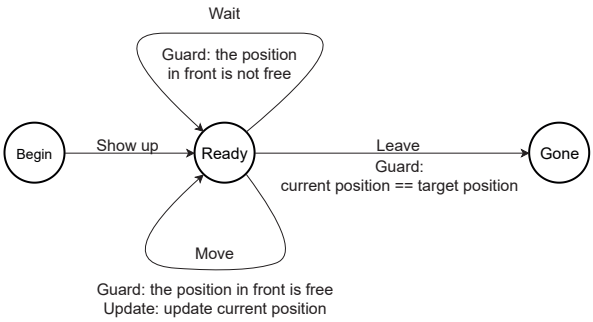


Figure 4: State diagram for *vehicle*

Finally, there is an additional textual description of a number of relevant aspects of the model. It mentions that a roundabout itself can be modeled as well while that is not strictly necessary since the *vehicles* know where they are through their *current position* property; that the *show up* action does not necessarily require a *vehicle* to stop in front of an empty roundabout, and that it is up to the modeler to decide on design details such as how far can a *vehicle move* during one run of the *go* procedure.

As elaborated in the section on assignment (section 5.3.1), the evidence the students provide about their models is twofold: the textual descriptions they formulate when designing the model and the implemented model, i.e. NetLogo program code (Müller et al., 2014). Our description format helps educators to distil the relevant aspects of the models from the descriptions and code students turn in and to employ the rubrics based on the SOLO categories.

With our format in mind, for the characterization of these SOLO categories we followed the local-to-global perspective. The first three levels describe quantitative progression. Prestructural indicates not answering the question at all or missing the point. Unistructural indicates fragmentary knowledge from a local perspective like mentioning only some of the relevant aspects — such as only a few agents or only some of their states — or missing important details. Multistructural indicates a more complete and coherent multi-point picture of the aspect under scrutiny — like listing all necessary agents and their states — but missing significant connections and without substantiating, clarifying, analyzing or explaining. These activities, however, are characteristic for the last two levels that add qualitative aspects: the relational level requires additionally the understanding of the relations among the parts of the aspect under scrutiny, such as recognizing all the actions an agent can perform, properly identifying conditions for these actions to occur and acknowledging their consequences. Finally, generalizing or theorizing — *what if?* — about aspects indicate going beyond what was given and typify the extended abstract level (Biggs & Tang, 2011; Meerbaum-Salant et al., 2013). In section 5.3.2, we elaborate the description of the SOLO levels for each criterium defining performance in detail.

5.3.1 Assignment

The assignment consists of a number of questions the students need to answer in writing while designing their model and using it to answer their research question, and of course, the task of implementing the model. After forming groups and choosing a case to model, the students answer the following questions:

Case and research question. Describe what you are going to model and with what purpose:

1. What do you know about this phenomenon? If need be, carry out the necessary research.
2. What part of your phenomenon would you like to build a model of?
3. What do you hope to observe from this model? (Questions 2 and 3 suggested by Wilensky and Rand (2015).)

Design the model. Design and describe the model following the questions listed here. Report the considerations and choices you make. (E.g., *"The sheep can reproduce. If two sheep meet, there is a chance of 20% that a new sheep will be breed. We decided not to take into account the gender of the sheep because that is not relevant in this case.")*

1. What are the principal types of agents involved in this phenomenon?
2. In what kind of environment do these agents operate? Are there environmental agents?
3. What properties do these agents have (describe by agent type)?
4. What actions (or behaviors) can these agents take (describe by agent type)?
5. How do these agents interact with this environment or each other?
6. If you had to define the phenomenon as discrete time steps, what events would occur in each time step, and in what order? (All questions suggested by Wilensky and Rand (2015).)

Implement the model. Implement the model in NetLogo. Write your code in small chunks and keep testing!

Validate the model.

1. Microvalidation: to what extent does the agents' behavior resemble the observations of the phenomenon in reality? If the behaviors are (somewhat) dissimilar, is this variation relevant to your research question?
2. Macrovalidation: to what extent does the behavior of the system as a whole resemble the observations of the phenomenon in reality? If the behavior is (somewhat) dissimilar, is this variation relevant to your research question?

Experiment, analysis and conclusion. Use the model to answer your research question:

1. Describe the experiment in detail. If you use Behavior Space, report the number of experiments conducted and the parameters used.
2. Report the findings in an appropriate manner (e.g., a narrative, a table, a graph, etc.)
3. Analyze the results.
4. Answer the research question.

Reflection. Reflect on your modeling process:

1. What went well and what could be better?
2. Did you make any assumptions which, in retrospect, you would like to reconsider?
3. Are there any aspects of your model which you would like to change? Are there any aspects of your model (agents or behavior) you decided not to include in you model while now you believe they do need to

be included? Make a wish list of aspect of your model that need to be added, removed or changed in the next version of the model.

Students were also asked to log their activities, problems, and successes; possible explanations for problems and successes, and, lessons learned.

5.3.2 Grading Rubrics

After we identified the criteria that defined performance, we created performance descriptions (Wolf & Stevens, 2007) to describe the appropriate level of understanding for intended learning outcomes (Biggs & Tang, 2011). Here we quote these descriptions:

Case and research question. (1) What do you know about this phenomenon? If need be, carry out the necessary research.

- *Prestructural:* Nothing or simplistic idea of the phenomenon. Performed no research.
- *Unistructural:* Some general description. Performed no research or only limited to isolated aspects of the phenomenon
- *Multistructural:* Performed some research. Able to name more relevant aspects of the phenomenon, but mentions no relations among these aspects
- *Relational:* Performed research. Complete idea of the phenomenon. Able to name relevant aspects of the phenomenon, have insight into relations among these aspects
- *Extended abstract:* Additionally, described the relation of this phenomenon to other phenomena in the world and/or conceptualized this phenomenon so as to be able to use it other contexts restricted and its relevance explained. Stated its relevance for other phenomena

(2) What part of your phenomenon would you like to build a model of?

- *Prestructural:* Nothing, or a few non-specific remarks but missing the point
- *Unistructural:* Few isolated aspects of the phenomenon identified
- *Multistructural:* Described what (part of the) phenomenon is being modeled.
- *Relational:* Clearly explained what (part of the) phenomenon is being modeled, together with limits of what is being modeled and its significance for the whole.

- *Extended abstract:* Additionally, theorize about possible generalization of the model or transfer into a different context.

(3) What do you hope to observe from this model?

- *Prestructural*: Research question not clear
- *Unistructural:* Identified the question from a local perspective
- *Multistructural:* Described the question from a multi-point perspective
- *Relational:* The research question clear and predicts possible outcomes.
- *Extended abstract:* Additionally, theorize about possible generalization or transfer into a different context.

Design the model and implement it.

- *Prestructural:* No agents mentioned
- *Unistructural:* A few agents and actions identified
- *Multistructural:* Several agents and actions described.
- *Relational:* Agents, actions and interactions correct and substantiated. Their contribution to the whole acknowledged.
- *Extended abstract:* Additionally, generalize or hypothesize about similar models in different contexts or extend the model beyond the minimal requirements.

Validate the model.

- *Prestructural*: Nothing. No working program.
- *Unistructural*: Identified some resemblances and differences between the model and reality. Relevance for the research question not clear.
- *Multistructural*: Described resemblances and differences between the model and reality. Relevance of the differences for the research question not clear
- *Relational*: Resemblances and differences between the model and reality described. Analyzed and explained their relevance for the research question.
- *Extended abstract*: Additionally, hypothesized over model adjustments to improve its validity for a more general purpose

Experiment.

- *Prestructural*: nothing
- *Unistructural*: a few model runs (i.e. simulations) without a clear plan
- *Multistructural*: Simulations performed systematically but the relevance for the research question not clear (e.g. not clear why certain data is gathered)

- *Relational*: Simulations performed systematically. The relevance for the research question is made clear.
- *Extended abstract*: Additionally: The relevance for the research question is clear and substantiated.

Analysis.

- *Prestructural*: nothing
- *Unistructural*: some results reported
- *Multistructural*: results described in an appropriate manner
- *Relational*: results described in an appropriate manner. The relation between the values of model parameters and the output data analyzed
- *Extended abstract*: Additionally, explain or hypothesize about the relation between the values of model parameters and the output data

Answer the research question.

- *Prestructural*: No answer
- *Unistructural*: Simple answer
- *Multistructural*: Elaborate answer, but the coherent picture not formed
- *Relational*: Elaborate answer, coherent picture of the parts and the whole formed
- *Extended abstract*: Additionally, discussion

Reflection.

- *Prestructural*: No answer
- *Unistructural*: Few aspects mentioned
- *Multistructural*: Several aspects described
- *Relational*: Aspects analyzed and explained
- *Extended abstract*: Additionally, discuss the possible consequences in the future

5.4 Method

5.4.6 Educational Context

Four classes participated in this study: one 11th grade VWO class and two 12th grade VWO classes which we all treated as one for the purpose of this research, and one 11th grade HAVO class. The data were collected in two schools in classes

that were taught by two teachers: the thesis author and her colleague who worked in both schools. All the students have previously learned to program in Python or a similar high-level textual programming language. The course on Computational Science lasted eight weeks in total. The first five weeks were dedicated to instruction using the teaching material we developed for our curriculum intervention, and during the last three weeks, the students formed groups of two or three (a few students choose to work alone) and worked on the practical assignment. After choosing the cases to model, the students went on to answer the questions form the assignment and to develop their models in NetLogo. The students from the two 12th grade VWO classes presented their models in the classroom and got feedback from other students. In other classes there was no opportunity to organize presentations. Finally, the students answered the last questions from the assignment and turned in their documentation and models, i.e., Netlogo code.

We analyzed only the completed projects turned in by sixteen students forming eight groups in 11th grade HAVO, fifteen students forming five groups in 11th grade VWO and twenty-four students forming twelve groups in 12th grade VWO.

5.4.7 Data Collection

Both the teachers assessed the students' work — project documentation and program code — using the rubrics presented here, assigning 0 up to 4 points for the prestructural up to extended abstract level, respectively, First, they separately assessed work of two groups. They compared their scores, and then the scoring guidelines and the interpretation of the rubrics were fine-tuned where necessary. Additionally, they agreed to take into account the answers students supplied while answering other questions — e.g., elaborating on the research of the phenomenon under scrutiny while answering the question about validation. Then, one teacher assessed the work of all the groups while the other teacher assessed only the work of the 12th grade VWO groups in order to establish the inter-rater agreement.

5.4.8 Analysis

Our aim was to investigate the inter-rater agreement and the discriminative validity of instrument. The inter-rater agreement was evaluated by computing Krippendorf's alpha for each of the criteria. Discriminative validity was assessed by comparing the scores of HAVO students and VWO students. While it is to be expected that both HAVO and VWO students can be taught to design and implement a model equally well, VWO are expected to outperform HAVO

students in defining and analyzing the consequences of manipulation of factors in problem contexts due to their ample preparation in scientific thinking. One-tailed t-tests for independent samples were used to evaluate these expectations. We expect that the majority of scores vary between score 0 (prestructural) and 3 (relational). If the projects of VWO students are more often awarded a score of 4 (extended abstract) than those of HAVO students, this too can be regarded as (partial) evidence that the instrument differentiates satisfactorily between students' level of learning outcomes. In order to see whether the students' results meet our expectations and in search of possible explanation for the observed differences in the performance between HAVO and VWO groups, we additionally performed qualitative in-depth analysis of the students' projects.

5.5 Results

All the results of the assessment using the grading rubrics are visually presented in figure 5 and figure 6, and in aggregate form as mean values of scores per school type (HAVO or VWO) per criterium in table 6, where the significance levels of the t-tests are presented too. In the figures, each row represents the results of one student group for the nine criteria assessed, and the height of each block represents the score, ranging from 0 for prestructural level to 4 for extended abstract level.

To assess the inter-rater agreement, we computed Krippendorf's alpha assuming ordinal scale from the scores of the twelve 12^{th} grade VWO project assessed by both teachers. The result is 0.78 which is a satisfactory value. Then, we used t-tests to find out whether the achievements of all the VWO groups taken together differ significantly from the achievements of the HAVO groups. We used a one-tailed t-test with significance level of 0.05. The results show that the groups differ significantly (printed bold in Table 6) for all the criteria except designing and implementing model, and for reflection.

We now examine the students' projects per criterium and per type of school, i.e. HAVO vs. VWO. We state our findings in general terms and illustrate them with characteristic text segments taken from the data.

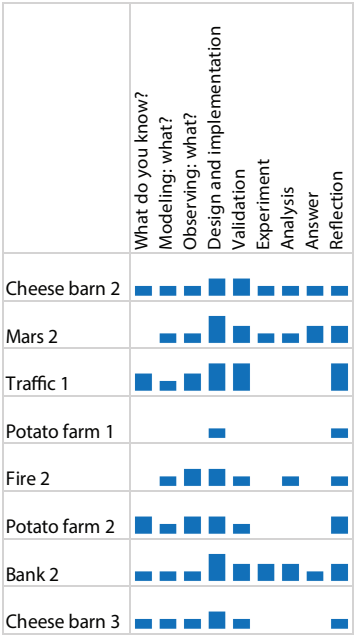


Figure 5: Cases and scores of the HAVO groups

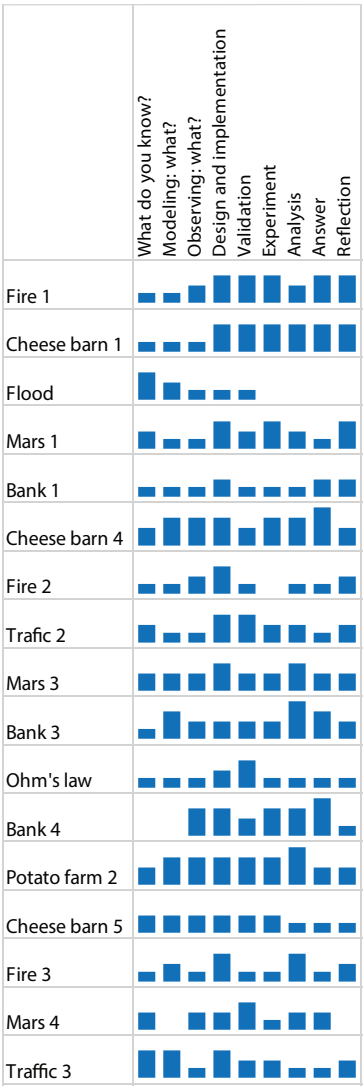


Figure 6: Cases and scores of the VWO groups

	What do you know?	Modeling: what?	Observing: what?	Design and implementation	Validation	Experiment	Analysis	Answer	Reflection
Total	1.38	1.43	1.68	2.43	2.08	1.58	1.64	1.66	1.77
HAVO	0.88	0.88	1.25	2.25	1.50	0.57	0.71	0.57	1.63
VWO	1.52	1.59	1.79	2.48	2.24	1.83	1.86	1.93	1.81
t-test significance	**0.031**	**0.031**	**0.049**	0.188	**0.012**	**0.002**	**0.007**	**0.002**	0.284

Table 6: Mean scores and significance levels of differences in the performance of the HAVO groups compared to the VWO groups.

Case and research question. While many groups — in HAVO as well as in VWO — did not engage in any form of research and relied on their existing knowledge of the phenomenon under scrutiny, we see significant differences between HAVO groups on one hand and VWO groups on the other. Among the HAVO groups, none rose above the quantitative levels of SOLO. For example, the group exploring life on Mars (case Mars 2), when answering what do they know about it, simply wrote, *"that life on Mars will not happen for a long time"*. However, this group made a good model and showed reasonable performance in the rest of the assignment. In the VWO groups, we saw a great variation with some groups reaching the relational level of SOLO by performing extensive research (e.g, looking up how much carbon dioxide does a person produce) and hypothesizing about possible outcomes of the model. For example, when describing what do they hope to observe form their model, the VWO group exploring the optimal strategy for potato farming (case Potato farm 2) wrote, "*we hope to find out what factors help increase the profit and what factors do not influence it that much. […] Out hypothesis is that fertilizer will really increase the profit but that pesticides have little influence on it. To prevent diseases and pests, we believe it is important to have large distance between the plants and to remove affected plants immediately.*" They continue in their logbook, "*We discussed what aspects of the phenomenon influence the profit. We decided not to take into account the seasons, weather and water. We did this because we look at the ideal conditions and other factors will not have a big influence.*"

Designing and implementing the model. For this part of the assignment, we see the highest scores achieved and small variation among the classes. Every group managed to design a model and more than half of them arrived at a model matching or exceeding our minimal expert model. A small number of groups, while succeeding in writing some code, did not manage to write a meaningful program and subsequently failed to use it to perform an experiment. An example is the VWO group working on case Flood who stated their case and research question but failed to implement their model properly.

Validate the model. Here we see a great variation in scores and the VWO groups outperforming the HAVO groups significantly. From the HAVO groups only one reached relational level and one group did nothing meaningful to validate their model. The response from the Cheese barn 3 group is exemplary: *"The agents simulate the production and sales of a cheese barn, thus the cheese production, the sale of cheese, the agents simulated the production and sale of cheese as it happens in the real world, so this corresponds well with each other.*" All the VWO groups

validated their models to some extent and almost a third of them reached the relational level, e.g. saying in case of life on Mars (case Mars 4), "*… maintenance of buildings and solar panels … are not relevant for the results of our model but could play a small role for our research question.*"

Experiment, analysis and answering the research question. Again, here we see a great variation in the scores with the VWO groups significantly outperforming the HAVO groups. While 15 out of 17 VWO groups performed an experiment and a number of them extensively employed the Behavior Space (a feature of NetLogo allowing for systematic parameter sweeping and recording the results of each model run), out of the eight HAVO groups only three provided evidence of performing an experiment and none of them used Behavior Space. The quality of the subsequent analysis of the results and the answers to the research question seem to be directly related to the quality of the experiment. Analysis and answering the research question are the only aspects of the assignment where a total of four VWO groups reached extended abstract level. The VWO group exploring the case of bank counters (case Bank 3) notes, "*assigning the tasks to specific counters leads to specialization of employees … this division of labor leads to faster and more efficient performance: by performing the same task all the time, the employees become specialized in particular tasks allowing them to carry out these tasks quicker. Conversely, non-specialized employees will carry out their tasks slower: because they get varying tasks all the time, they lack specific knowledge and need to look up things for the customers, causing the task to last longer. Because of this, tasks in scenarios 2 and 3 would take longer. However, the question is to what extent does this counterbalance the efficient engagement of the bank counters. This, then, is something that would require further research.*"

Reflection. The HAVO groups and VWO groups perform similarly. Interestingly, none of the groups in the 12th grade VWO reached relational or extended abstract level. One of the 11th grade VWO groups reaching that relational level explored evacuation of a burning building (case Fire 1) and wrote in their wish list, "*What we'd like to implement in our model is turtles making a clear choice what emergency exit to take. … Sometimes they seem to doubt what exit to take, walk back and forth between the exits and eventually wait for too long — dead through fire. That's a pity because in reality, they could've survived.*"

Logbooks. In one of the schools the students were asked to keep a logbook, and in the other they were not, so the logbooks were not assessed separately. However, following the assessment finetuning guidelines, we read the submitted logbooks looking for evidence of answers to other questions.

5.6 Discussion and Conclusion

We designed and investigated an assessment instrument for the assessment of the intended learning outcomes for Computational Science. The design process was all but straightforward due to the fact that some ILO's of modeling are at the core of CS (e.g. implementation of the model), while others are not often seen in a CS classroom (e.g. experiment). Even for implementation, which comes down to programming, it was not easy to find related work addressing assessment of programs at just the right level of abstraction. The same holds true for validation: while there is plentiful literature on validation of computational models, we could not find any focusing on the assessment of validation in a formal learning setting.

The project documentation and program code proved to be sufficient sources for assessment. However, when possible, we suggest to let students present their projects in the classroom too, and we encourage the teachers using this assessment instrument to take into account their observations of students at work when assessing their projects, as suggested by a number of teachers participating in one of our previous studies (Grgurina et al., 2017). Indeed, the teacher who cooperated in this research noted that, while assessing his students' work, he constantly thought of his impressions from the classroom and wanted to take these impressions into account. This might be especially important for students who perform poorly when verbalizing their thoughts, as witnessed with many HAVO students in the parts of the assignment requiring textual descriptions such as stating the case and research question. We saw that none of these students achieved extended abstract level, while in the 12th grade VWO one teacher found four instances of student groups reaching it. The other teacher, however — while assessing the same projects — found none and said, *"it was difficult to see clearly where the boundary lies between relational and extended abstract levels."* Therefore, it could be argued that the extended abstract level is unobtainable for HAVO students, which would signify a situation similar to the one described by Castro and Fisler who found no instances of extended abstract level in their students' work (2017). Meerbaum-Salant et al. (2013) did not consider it at all and designed assessment with only the three intermediate SOLO categories to monitor novices' learning of CS concepts. An issue to consider here is the question, what level of understanding is intended for the HAVO students, as opposed to the VWO students, and with what purpose are the students learning about Computational Science. The HAVO students are following education stressing a

hands-on approach and leading to higher professional education and are often described as *thinking actors* (Barendsen & Tolboom, 2016), which could explain why there is no significant difference in their performance levels — compared to those of the VWO students — when designing and implementing models. The VWO students — often described as *acting thinkers* (Barendsen & Tolboom, 2016) — prepare for universities in an educational setting embracing a scientific frame of mind and it is not surprising that they significantly outperform HAVO students when validating their models and using them to conduct research — i.e. perform experiments, analyze results and draw conclusion. Therefore, we want to encourage teachers using this instrument to put more emphasis on the aspects of the modeling process related to the specific needs of their students. Arguably, for the HAVO students it might be more important to get a clear picture of the phenomenon being modelled and focus on the development — or possibly only enhancement — of a model, while for the VWO students, with the whole of their education emphasizing the scientific attitude, it might be more important to view a model as a vehicle to engage in scientific research and develop and use it as such. In order to cater to their needs, we repeat our recommendation to further sharpen the instruction about experimentation and data analysis (Grgurina et al., 2016) and add a suggestion to actively coach students in the first phases of their modeling projects when stating the case and research question and performing the accompanying research.

In conclusion, our assessment instrument in the form of a practical assignment and accompanying rubrics based on the SOLO taxonomy proved to be reliable, as indicated by a high rate of inter-rater agreement. Its validity is corroborated by exposing the significant differences in the performance levels of the HAVO students compared to the VWO students: as expected, the performance levels of the VWO students were significantly higher for almost all the criteria.

The results of this study exposed the needs of specific groups of students to receive instruction prior to and during their work on the assignments, and they informed us about the shortcomings in the curriculum intervention. All of these findings will contribute to the further refinement of the instrument itself, to the development of the teaching materials — an effort that will be reported elsewhere — and to the development of the CS curriculum in secondary education in the Netherlands, CS teacher training and CS education in general.

Chapter 6

Modeling and Simulation: Students' Understanding and Difficulties Related to Verification and Validation

In this final study, we focus on students' understanding and difficulties while working on Computational Science assignments in CS class. We interviewed eleven 12^{th} grade secondary education students who made models and ran simulations of phenomena from other disciplines, and we charted their understanding related to modeling aspects research, abstraction, verification/validation and reflection together with difficulties they experience. We saw that the initial phases of the modeling cycle did not represent great challenge to the students, but that they still experience difficulties in the abstraction and verification/validation phases. The results of this study will inform the further development of the teaching materials, teaching methods and teacher training.

6.1 Introduction

In this final study of our project, we focus on students' understanding and difficulties while working on Computational Science assignments.

In our first study (chapter 3) — focused on Magnusson's component M1 concerned with the goals and objectives as well as curricula related to a specific topic being taught — we obtained an operational description of the learning objective Computational Science that describes the modeling cycle for simulation modeling through its elements *purpose*, *research*, *abstraction*, *formulation*, *requirements/specification*, *implementation, verification/validation, experiment*, *analysis*, *reflection*. We also identified specific challenges the students experience when engaging in modeling activities: difficulties to decide what phenomenon to model, to determine the right level of abstraction, and to formulate the problem under scrutiny suitably for modeling through ABM; inadequate programming knowledge, inability to correctly attribute unexpected program behavior to either errors in the model itself or to emergent behavior of the model, and finally, not conducting experiments with their models in a systematic fashion. In our second study (chapter 4), we explored computer science teachers' initial PCK on modeling and simulation and in particular, their ideas about instructional approach to teaching Computational Science and assessment. The finding of these two studies informed the development of our teaching materials, a practical assignment and an accompanying assessment instrument. In our third study (chapter 5), we focused on the development of that assessment instrument to monitor the levels of understanding in the learning outcomes of students engaging in modeling projects, thus focusing on Magnusson's component M4.

In this study, we focus on students' understanding and difficulties while working on Computational Science assignments using the teaching materials we developed — Magnusson's component M2. Our aim is to explore these, with a particular focus on their actions leading to the development of valid models. We seek to answer the following research question:

1. How can the students' understanding of model validation be portrayed in terms of validation techniques they employ to ensure the development of valid models?
2. What difficulties do the students encounter when verifying and validating their models?

Our findings will serve to complete our exploration of teaching Computational Science in the context of the CS course in Dutch secondary education and they will contribute to the further development of teaching materials and teacher training courses.

6.2 Theoretical Background and Related Work

6.2.1 Modeling Cycle

Building and using a simulation model is an iterative, cyclic process consisting of five stages and reflection on modeling process, as described in detail in chapter 3.

In the definition stage, the intended *purpose* of a model is stated and the modelers perform *research* about the phenomenon that is being modeled.

In the conceptualization stage, in the process of *abstraction* it is decided what aspects to include in the model and what to leave out, and the problem is *formulated* in a way that enables us to use a computer and other tools to help solve them (CSTA Computational Thinking Task Force, 2011). This leads to the development of a conceptual model which is validated to ensure *conceptual validity* of a model. During this validation step, it is determined whether the conceptual model is built upon *correct theories and assumptions* (Sargent, 2013) and the modeler can consult domain experts and the customer to ensure this aspect (Law, 2009). If necessary, these steps are repeated until the conceptual model is satisfactory (Sargent, 2013).

In the subsequent formalization stage, a conceptual model is implemented — i.e. programmed using software engineering techniques — yielding a simulation model which is *verified* to make sure the programming is correct, i.e. ensuring that it really does what the modeler think it is doing (Brade, 2004; Sargent, 2013; Sturrock, 2015). Again, this step — and possibly adjusting the conceptual model too — can be repeated until a satisfactory simulation model is obtained (Law, 2015). This simulation model is then subject to *operational validation* to determine whether the model's output behavior has a satisfactory range of accuracy (Sargent, 2013). The results of the *operational validation* process can prompt adjustments of the conceptual model or the implemented simulation model, and in that case, these steps can be repeated until a satisfactory simulation model is obtained.

In the execution stage, the model is used for its purpose, i.e. to design and run *experiments*.

Finally, in the conclusion stage, the results of the execution stage are *analyzed* and translated back into the problem domain.

Following each of these stages and upon the completion of the modeling process, reflection takes place and the whole process is possibly repeated.

6.2.2 Validity

While it is impossible to make a perfect model (Rand & Wilensky, 2006; Sturrock, 2015), there are many *validation* approaches and techniques ensuring that the produced model is accurate, credible, and fit for the intended purpose (Law, 2009).

The measures modelers can take to contribute to the development of valid models include the model construction, testing the models, enlisting the help of others and reflection about the models, as illustrated in the figures 7.

Figure 7: Validating a model

During the iterative process of the model development, the construction and testing of a model are intertwined. Models are constructed upon assumptions (Sargent, 2013) resulting from *research* into the existing theories and possibly historical data concerning the phenomenon to be modelled, and *abstraction* — as described in the section on the modeling cycle (see figure 8). A model can be *calibrated* in order for it to be *in harmony with the real world* (Carley, 1996).

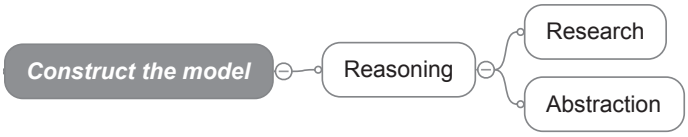


Figure 8: Constructing a model

On one hand, the validity of a model can be derived by having confidence in on validity by construction, thus relying on *philosophy of science — rationalism* approach which entails that a model is correctly developed from clearly stated reasoning (Sargent, 2013).

On the other hand, numerous validation techniques rely on *testing* to judge the validity of a constructed model depending on its behavior and outcomes that are *generated* as the input parameters are varied, and then *observed* and *interpreted*.

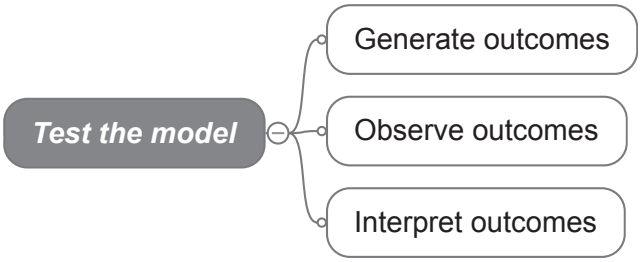


Figure 9: Testing a model

To *generate* the outcomes, several tests can be performed. Examples of such test are *stress test (i.e testing with wide range of parameters and random numbers)* (Carson & John, 2004), *parameter variability–sensitivity analysis (i.e. varying the values of parameters and observing the resulting outcomes to determine whether the relations correspond to those of the real system), extreme condition test (i.e. checking that the model's structure and outputs are plausible for extreme and unlikely combinations of the system's levels of factors) and degenerate tests (i.e. looking into the degeneracy of the model's behavior)* (Sargent, 2013).

To *observe* the generated outcomes, several methods can be employed. The generated outcomes can be observed visually through *animation (i.e. graphical display of the model's behavior as it is run)* and *trace (i.e observation of the behavior of a specific entity during the run of the model) (Sargent, 2013).* Another way to observe the generated outcomes is by making use of *quantitative measures* expressed through *operational graphics (i.e. graphical display of various performance measures as the model runs)* (Sargent, 2013) or numerically through various *performance measure*s (Law, 2009).

Finally, the outcomes are *interpreted*. Several techniques can be used to interpret the model's outcomes. Examples of such techniques are comparison with real data through *historical data validation* or *predictive validation* and checking for *event validity (i.e. comparing the 'events' occurring during the model run with those occurring in the real system)*, checking *input-output consistency* and *data relationship correctness (i.e. making sure "Data relationship correctness requires data to have the proper values regarding relationships that occur within a type of data,*

and between and among different types of data.") (Sargent, 2013). Furthermore, a model can be *compared to other models* (Bungartz et al., 2014; Wilensky & Rand, 2015), and finally, its *consistency with previously verified theories* can be checked (Bungartz et al., 2014). Notably, when a real system does not exist, a model can still be considered valid while not accurate (Schmid, 2005).

When interpreting the model's outcomes, the judgement about the model's validity can be based on objective criteria that use statistical tests on the outcomes of a model (Law, 2009; Sargent, 2013), or on subjective criteria.

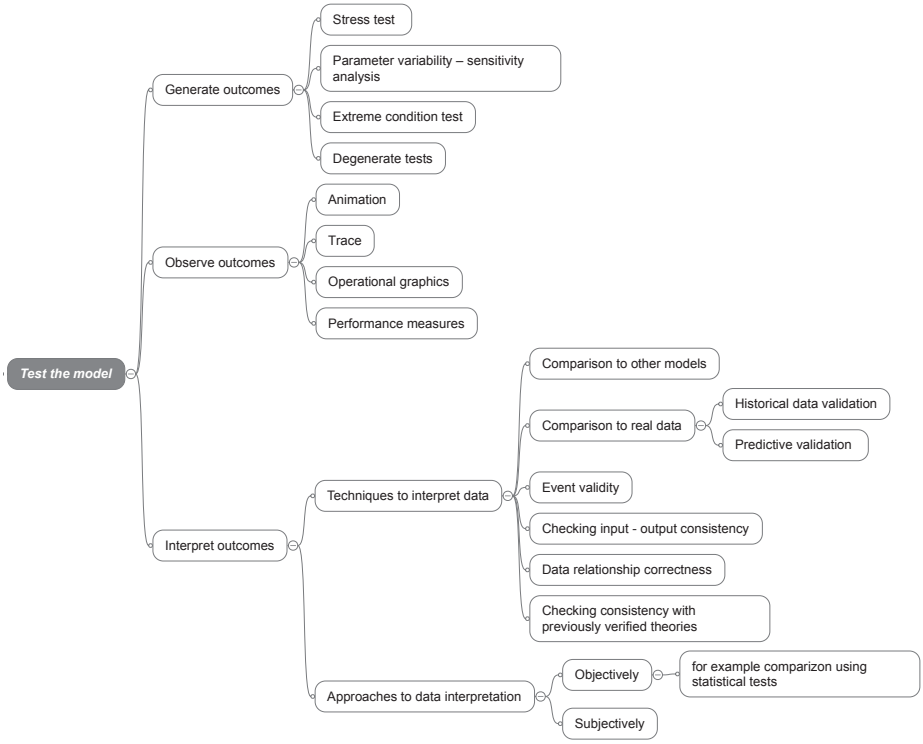


Figure 10: Testing a model — various techniques

In addition to these techniques, the modelers can involve various stakeholders to support the validation process. A peer group can be involved to assess the model's correctness through a *structured walkthrough* (Law, 2009; Sargent, 2013). The customer ordering the model can be asked to participate in the *review of the model* (Carson & John, 2004) or domain experts can be consulted to assess the *face validity* of a model (Sargent, 2013; Wilensky & Rand, 2015) through, for example, *Delphi test* involving a panel of experts (Carley, 1996) or *Turing test* where experts

are asked whether they can distinguish the model's outcomes from the outcomes of the real system (Sargent, 2013).

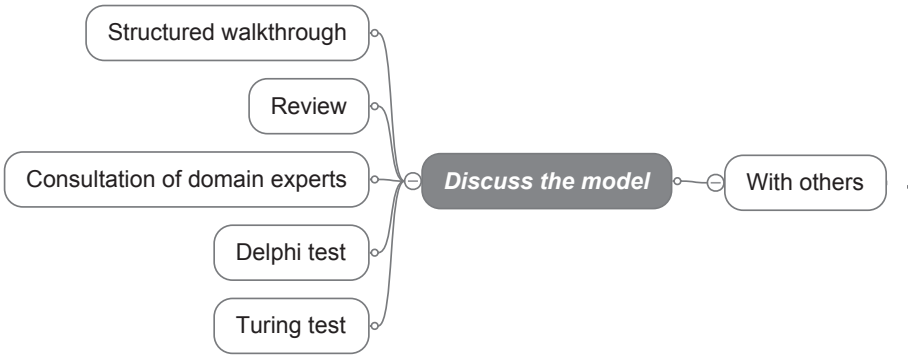


Figure 11: Validating a model with others

Reflecting on the model takes place alongside with technical aspects of validation, establishing the *degree of confirmation* (Naylor & Finger, 1967) — and similarly, *accuracy* (Schmid, 2005) — by determining *satisfaction* that the desired results were sufficiently achieved (Bungartz et al., 2014) and *credibility* — i.e. degree of confidence in the model (Brade, 2004) — play a crucial role. Both the modeler and the customer who ordered the model need to be convinced that the model is *plausible* — i.e. its results do not contradict previously validated theories — and sufficiently fit for purpose (Brade, 2004; Law, 2009; Sargent, 2013).

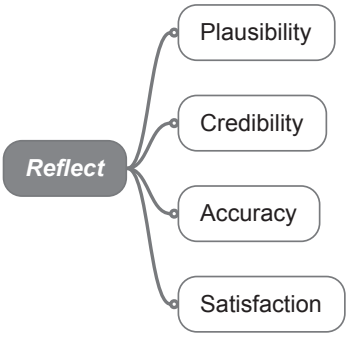


Figure 12: Reflecting on a model

As mentioned in the section on the modeling cycle, any of these actions can be repeated until a sufficiently valid model is obtained.

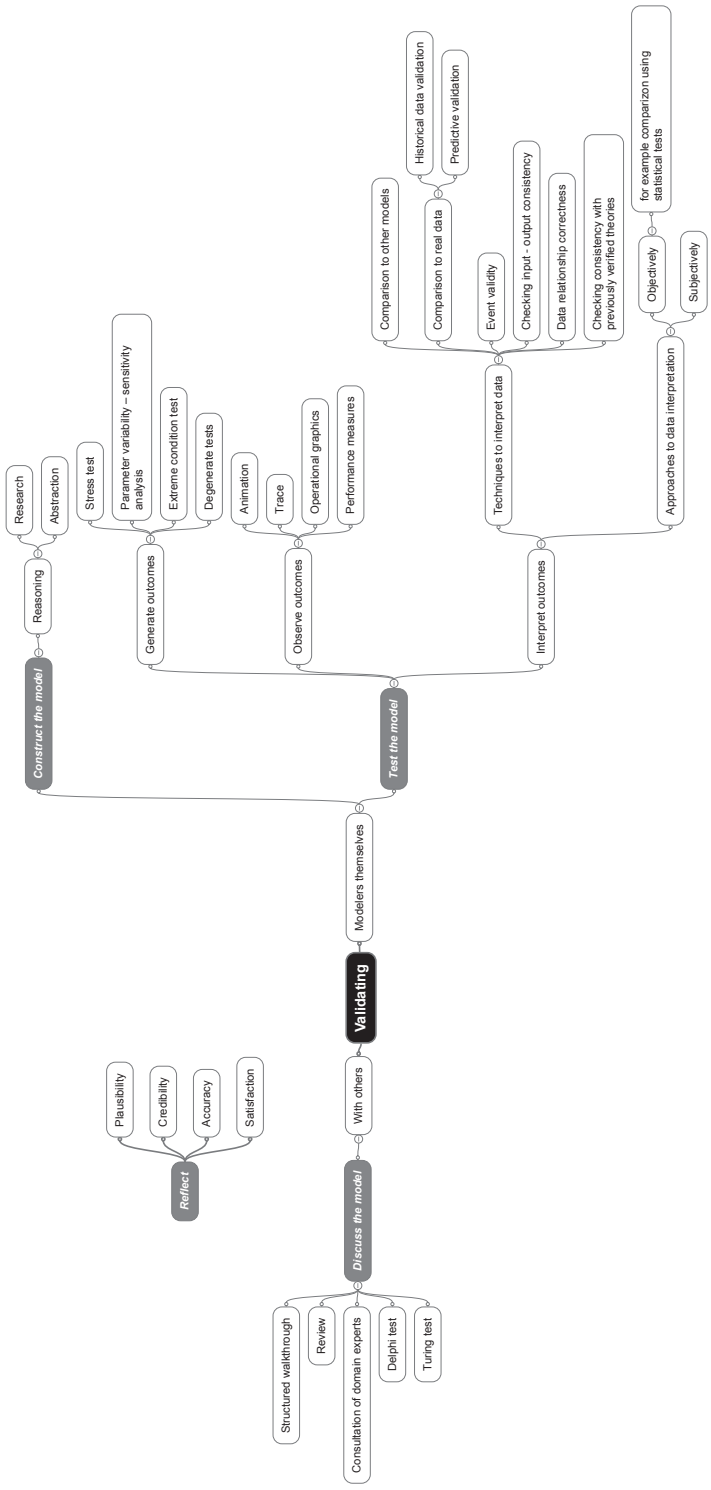


Figure 13: Coding categories

6.3 Method

6.3.1 Educational Setting

The study took place within the regular CS course the first author teaches to 12th grade students. For them, this was the last year of their three-year CS course. During an eight-week period they studied Modeling and Simulations with NetLogo using newly developed teaching materials. The first five weeks were dedicated to studying the teaching material and making the accompanying assignments. During the rest of the period, the students worked in small groups (mostly in pairs) on a large practical assignment where they investigated various phenomena by making a model in NetLogo and exploring it through running simulations. This process was strictly planed and contained milestones when the students produced the required project documentation and kept logbooks. At scheduled moments, the students handed in all the documentation through google forms. In these forms, they had to describe their phases of the modeling process and in particular, they were asked to elaborate on the validation they performed. This practical assignment is detailed in chapter 5. At the end of the period, each group presented its model to the rest of the class and the students were encouraged to discuss their models, results, design choices, programming issues and other questions they found relevant. A few days later the students turned in their final reports and NetLogo programs. They were encouraged to improve their models taking into account the feedback they got during the presentations.

6.3.2 Participants

Out of the twelve students in the class, one opted out and decided not to participate in the study. The eleven participating students — two girls and nine boys — formed six groups. The overview below shows the six groups (G1 through G6) consisting of students S1 through S11 and the cases they worked on.

- G1 (S1 and S2): Potatoes. A potato farmer wants to maximize the profit while taking into account the costs for seed plants and fertilizer, contagious diseases and pests whose propagation depends on the distance among the plants and the engagement in clearing out affected plants. What is the optimal strategy for planting and farming potatoes?

- G2 (S3 and S4): Fire evacuation. In case of fire, it is important that people are alarmed quickly in order to leave the building on time. What are the optimal numbers of alarm bells and emergency exits on the ground floor of your school to achieve this goal?
- G3 (S5 and S6): Life on Mars. After the initial supplies and shelter are delivered to Mars, would it be possible to produce sufficient water, air and food to survive? In other words, would be possible to found a sustainable human colony on Mars?
- G4 (S7 and S8): Ohm's Law. Check the Ohm's law for series circuit and parallel circuit.
- G5 (S9 and S10): Cheese barn: A cheese farmer produces cheese and stores it in a warehouse with limited capacity. The older the cheese, the high the price it fetches. The farmer can sell the cheese to supermarkets with guaranteed purchase and fixed price, or at the farmers market for higher price but uncertain sales volume. What are the optimal production and selling strategies?
- G6 (S11): Bank counters. Investigate the optimal number and task division of bank counters as to minimize the waiting time of the customers with various needs.

The purpose of the most of these models was to find an optimal solution for a problem through experimentation. Only one of the models — Ohm's Law — had a different purpose, namely, to seek explanation for Ohm's Law as a result of behavior of electrons.

6.3.3 Data Collection

After the project were graded, these eleven students were interviewed. We performed individual semi-structured interviews with them using the protocol cited below. While we were primarily interested in verification and validation, for the sake of a naturally flowing conversation and completeness, we enquired about the entire modeling process and emphasized the part concerning verification and validation. To help recollection during interviews, we ran students' models on a computer and presented their documentation.

Here we cite the interview protocol we used.

- Introduction, stating the purpose of the interviews
- Showing the students the documentation and models they turned in
- Discussing their modeling process through the following questions:

1. What phenomenon did you model?
2. What did you know about this phenomenon beforehand? Have you performed any research and what were your findings?
3. What did you hope to find out using this model?
4. How did you test your model?
   - 4.1. What did you do to debug and verify your model (i.e., your NetLogo program)?
   - 4.2. Validation
     - 4.2.1. How did you perform microvalidation to validate the behavior of individual agents?
     - 4.2.2. How did you perform macrovalidation to validate the behavior of the entire system?
   - 4.3. Was this a linear or a cyclic process? What did you encounter? Have you been changing code to achieve "better" behavior of the model?
   - 4.4. On what basis did you conclude that your model was valid? (note to interviewer: pay attention to: criteria formulated in advance, consultation with an expert and common sense.)
   - 4.5. To what degree are you convinced?
   - 4.6. To what degree are you satisfied with the validity?
5. Were you able to perform experiments with this model?
6. Did you obtain useful results?
7. Are you satisfied?
8. What is you wish list for a next version of your model?
9. Please reflect on the whole modeling process you engaged in.

The interviews were recorded and transcribed verbatim and the transcripts were used for data analysis.

6.3.4 Data Analysis

We performed a qualitative analysis of the interviews and of the documentation students turned in. We first identified text segments of documentation and interview transcripts in which model validity played a role. The units of analysis were coherent chunks of text pertaining to a specific action related to aspect contributing to determining validity, and reasoning behind it.

In the first analysis phase, the text segments were classified according to the core categories in our validation process model (see Figure 13):

- Construct the model through reasoning (by the modelers themselves)
- Test the model (by the modelers themselves)
- Discuss the model (with others)
- Reflect

In the second phase, we analyzed the strategies employed by the students within each category by means of an open coding process. In subsequent coding cycles, we grouped the descriptions of the students' validation activities into more abstract descriptive categories, using the elements of our validation process model (see white ovals in Figure 13) whenever possible.

In the third phase we looked for students' difficulties and misconceptions within each of the resulting categories and classified them in a similar open coding process.

In the process of determining the coding schemes for strategies and difficulties, the first and second author analyzed parts of the interviews separately and discussed their findings until they agreed on the classification.

6.4 Results

In this section, we portray the activities the students reported employing in order to develop validated models while constructing and testing their models, and while reflecting. We use our theoretical framework depicted in figure 13 to organize the presentation of the results. We state our findings in general terms and illustrate them with characteristic text segments taken from the data.

6.4.5 Constructing the Model

Models need to be built upon correct theoretical assumptions in order to make them *conceptually valid*, and, ideally, *calibrated* to fit the empirical data which are adequate and correct, thus ensuring *data validity*. To this end, one needs to possess adequate understanding of the phenomenon under scrutiny and engage in the process of abstraction.

Research

Adequate understanding of the phenomenon under scrutiny can be obtained through active engagement in *research*, as some of the students did, or by other means such as exploiting their own experience or utilizing what they learned at school.

In total, students reported relying on three sources of their knowledge of the phenomena they modeled.

1. *Experience*. All students rely on their experience or what they would consider common sense to a certain degree when considering their understanding of the phenomenon under scrutiny. Some of them consider this to be sufficient and feel no need to do extensive additional research, such as group G2 and presumably G6. Student S3 reported, "well, we had a fire drill once, bur further than that, we didn't know the science behind it" but added that they consulted an article on how to model fire drills. Student S11 did not report doing any research or using any form of real-world data.
2. *Learning at school*. With several of the phenomena modeled, students involve what they learned at school within some other school subject into their modeling either to inform the essential assumptions, or for some additional detail. Group G4, who modeled Ohm's law, relied on their existing knowledge of chemistry and physics and did not perform any additional research.
3. *Research*. In addition to their own experience or knowledge from school, some students go to great lengths when researching the data necessary for their model, and they subsequently *calibrate* their models to fit the data they found. Yet, this still does not guarantee a realistic model since some data are hard to find or it does not occur to students they need them. Groups G1 and G3 performed extensive research about the phenomena thy modeled and group G5 looked

up some of the data. Group G1 researched how many tubers a potato plant yields, what are the costs of fertilizer and spray, and other relevant data for potato farming. However, when asked how come their results suggested not to use fertilizer and spray, S1 responded that it was difficult to find out how much money a potato earns, and how much does it cost to fertilize and spray a single potato plant. Similarly, group G3 who modeled sustainability of human colony on Mars, investigated how much electricity does a solar panel produce, how much oxygen and tubers a potato plant produces, how much energy, oxygen and water a person needs to survive, how photosynthesis works, and many other relevant details. Student S6 comments, "we simply looked up many things the way they would resemble reality." On the other hand, group G5 did look up the categorization of cheese depending on age (young, mature, etc.) but did not investigate specific economic aspects of cheese production and sales. Rather, they relied on what they learned in the economy class about market forces in general.

Abstraction

All students pay careful attention to build their models upon correct *theoretical assumptions* and thus engage in the process of *abstraction* (Grgurina et al., 2016) where they decide which aspects of the phenomenon under scrutiny to take into account and in what form, and which ones to leave out. When employing a proper perspective, students use sound judgement and make appropriate choices concerning the theoretical assumptions underlying their models. This paves the road to building a model that is sufficiently fit for its purpose. For example, in the process of *abstraction*, group G1 chose to unite all possible potato diseases, infestations etc. into one phenomenon and call it pests. This decision simplifies the design of their model by disposing of the unnecessary details while retaining the essence of infectious and transferable diseases and infestations.

Among the six groups observed, we see three erroneous perspectives when performing *abstraction*.

1. *Oversimplification* occurs when the students go too far in their quest for feasible assumptions, leading to a very specialized model of limited usefulness. This is a conscious course of action caused by inability to design or implement a model that would be based

on more general and not so specific assumptions. Group G2 tried to make a model for evacuation of a burning building in general, thus with any floor plan. In their initial attempts, the agents in their model were walking through the walls on their way to the nearest emergency exit and the students did not know how to program the model to alleviate this behavior. Instead, they decided to use only a particular floor plan in combination with additional measures to prevent this problem from occurring.

2. *Omission* occurs when students are not aware there are aspects of the phenomenon under scrutiny that are essential in order for the model to represent that phenomenon correctly and thus make the model fit for its intended purpose. An example of such an omission is given by group G5 who modeled a cheese barn. In their model, the cheeses only get the opportunity to ripen if they do not get sold first, whereas in this case it is essential to put cheeses aside and let them wait until reaching their intended age before being sold.
3. *Circular reasoning* is a misguided attempt by a modeler to use the outcomes of a model as model's assumptions. Similarly to *omissions*, students are not aware of this logical fallacy when they employ it as a solution to the problem for which they have no idea how to solve otherwise. Group G4 unwittingly used Ohm's law to prove it. When the teacher pointed this out, student S7 commented, "and that was purely because at the moment we had no idea how we'd calculate the resistance and it didn't occur to us, like, you can't use that at all" and went on to conclude, "the result is that our model works like the Ohm's law, but it doesn't confirm it."

6.4.6 Testing the Model

Once the assumptions are established as a result of the research and abstraction process, the model is developed and implemented, i.e., programmed. In the process of *verification*, the modeler ensures that the model really does what they think it is doing (Sturrock, 2015). The simple fact that a program runs, i.e. there are no syntax errors, is no proof of a correct implementation.

In this section we report on what students saw as errors themselves and in the next section we report on what techniques they employed to diagnose those errors.

Students report building their models iteratively in small steps and running them all the time to test them, intertwining verification with validation (Wilensky & Rand, 2015). They report two types of errors encountered:

1. *Data omissions* happen when students erroneously copy or implement constants from their research data into their programs. Student S2 reported their program produced unexpected and improbable fluctuations of the number of people in their Mars colony. The error was caused by the fact that, in an early version of their program, a tick represented a day in the life of a person and a month in the life of a potato plant. When this error was fixed, the number of people showed smaller, acceptable fluctuations.
2. *Process omissions* occur when assumptions underlaying a model lack sufficient detail. For example, student S5, when modeling Ohm's law, observed in an early version of the model that the electrons "become stuck and could not go any further. I don't think it works like this, so we had to change it." They added a random component to the angle at which the electrons bounced from the atoms and that solved this problem.

If a model is to be useful, it needs to be valid. Here we describe techniques students employed to validate their models and report on measures they took to improve their models' validity.

During an iterative cyclic process, the students used a twofold approach to establish the validity of their models. They relied on the correct construction of their models from appropriate assumptions and then *reasoned* about their models to draw conclusion about the validity of the models, or alternatively, they *tested* their models: they *generated* model outcomes by varying the input parameters of their models, *observed* and *interpreted* the behavior of their models.

Generating Outcomes

All the students who tested their models engaged in *parameter sweeping* — a technique where the model's parameters are systematically varied to *generate* outcomes, and they subsequently *observed* and *interpreted* the outcomes.

Varying the parameters as a validation technique serves two purposes: to determine the influence of various parameter values to the model's output (*parameter variability - sensitivity analysis*) and to determine whether the

model's outputs fall into acceptable range (*operational validation*). The *parameter variability - sensitivity analysis* technique was employed by groups G1 and G2. For example, student S2, who modeled potato farming, when asked if their group was playing with the distance between potato plants, answered, "yes, that is the density percentage. How many plants are being planted." Almost all the groups employed *operational validation* to make sure their models' outputs fall into acceptable range. Student S4, modeling the evacuation of people in a burning building, said, "Alarm bell number 5 was in the middle, so then you had alarm bells, four emergency exits, that was I believe almost the best. […] was around 600 [tics], everything was around 600, that was good. Except, one alarm bell [only], that is alarm bell number one, I believe it was down there in the left corner. Well, then it takes really 900 tics."

The students we interviewed employed the *parameter sweeping* techniques not only to validate their models by getting confirmation that the models were good enough, but also to *calibrate* their models by choosing the appropriate parameter values, and during experiments which some of them saw as additional confirmation of the validity of their models as well. Groups G1 and G5 are examples of groups who employed systematic parameter sweeping while running their models to perform experiments and reported that the results thus obtained additionally convinced them of the validity of their models. So did group G3 too: they tested various parameter values and discovered that their Mars colony was sustainable when it contained 17 people. Group G4 also reported finding parameter values that guaranteed a desired constant output value.

Several students report *hard coding* some of the values into their models rather than having the user determine these parameter values when running the model. For example, the model made by group G6 has no input parameters at all, but the model's parameters are determined at random during the execution of the program.

Observing Outcomes

As the outcomes are *generated*, there are two main ways to *observe* the outcomes and behavior of a model: *visual inspection* and making use of quantitative *performance measures*.

Visual Inspection. We saw that almost all the groups employed *visual inspection* either by looking at the *animation* of the whole model, or by *tracing*

some of the agents. Consequently, some students were content with what they observed, while others realized their models needed to be improved.

Some students *observed* model's behavior, established *event validity* and *interpreted* the model's outcomes to be in line with reality and thus acceptable. For example, student S2, who modeled potato farming, said, "And that is what we looked at […] that a plant doesn't get sick at a random spot, but that it really gets infected [by the neighbors]". Student S11 *traced* the behavior of the individual agents waiting in the queue as well as *observed* the lengths of the queues through *animation* and also concluded that the model's behavior was realistic enough.

Other students encountered what they experienced as unrealistic behavior of their models. We observed twofold approach to dealing with this phenomenon: either fixing the problem or deciding that the problem is not relevant. Student S8, who modeled Ohm's law, saw an error concerning electrons not bouncing back when colliding with atoms, and fixed it, as described in the section on verification.

Group G2, who modeled the evacuation of a burning building, partly fixed the problem, as described in the section on abstraction (section 6.4.1). After fixing the problem, they observed that people in their model left the building quickly enough, so they decided that their model was nevertheless realistic enough.

Two groups did not employ visual inspection: in the cheese barn model made by group 5, visual aspects such as position, movement or interactions of cheeses play no role and cheeses only change color to indicate their age. Similarly, group G3 modeled no visual interactions in their Mars colony model and therefore relied on *observation* of *quantitative measures* to validate their model.

Quantitative Measures. Almost all the students *observe* the outcomes of their models through quantitative measures. To this end, they observe relevant numerical values produced by the model by keeping an eye on monitors[23] to observe *performance measures,* or charts produced while the model runs to observe *operational graphics* — and thus interpret the model's outcomes *subjectively*. In their report, group G1 who modeled potato farming, said, "We tested these things extensively by using monitors." Student S5, who modeled sustainability of human colony on Mars, said," we got a very oscillating line of the number of people on the planet, say, because there were a lot of people being born and, say, if there was enough food, then the potatoes were gone, then everyone died." Student S10, who modeled cheese barn, said, "… so you can never have more than 144, […]

23 A monitor in NetLogo is an interface element displaying the value of a variable.

144 cheeses are what, that you could see in the chart." Interestingly, student S11, who modeled bank counter queues, made extensive use of monitors in the model to observe its behavior and outcomes, but does not explicitly refer to them when asked about validating the model.

Interpreting Outcomes

In order to assess the validity of a model, the *observed* outcomes can be *interpreted*, either subjectively or objectively — for example using statistical tests.

When possible, the model's outcomes can be compared to the behavior of the real system.

Three groups reported comparing the events occurring in their model to those of a real system as they perceive it, thus employing *event validity* technique and *checking for consistency with previously verified theories*. However, they made no use of *historical data*. When asked what they think would happen in reality if the farmer did not intervene, student S1 replied, "I think the affected plants would take over. And here, the new, healthy plats are added all the time" and later went on to add, "affected plants die and in reality, a new plant would not be planted there. So it is, wait first for everything to be harvested and only then will the new plants be added." When the model results indicated that use of fertilizer made no difference, student S2 commented, "right, that is not realistic, so we knew through our common knowledge […] that the outcomes were not good." Yet, this group found their model sufficiently valid, as described in the section on plausibility and accuracy. Group G2 compared the behavior of their model with their own experiences of a fire drill at school and concluded that the model was satisfactory. Student S11 commented, "In reality people enter the building and join the queue, the shortest queue."

Reasoning about the Model

When the data from the real system are not available, validity of a model can be derived through reasoning. Here, one asserts that if the assumptions and the implementation of a model are correct, then it follows that the model is valid. This is what group G5 did and student S9 said, "so then we looked if this and this happened one after another, does that happen one after another in reality too? If that's right, then it should be right in the model too, because, say, we had no real information if it was really right, so no real information from cheese producers that you could fill in, like, are the outcomes the same."

6.4.7 Reflection

The validity of a model is related to its intended purpose. The modelers make subjective judgements expressing how *plausible*, *accurate and credible* they perceive their models and whether they are *satisfied* with them.

While some students go to great lengths to ensure their models are built upon realistic data and sufficiently *calibrated*, checking their models' outcomes against realistic data remains a problem.

Groups G1, G5 and G6 are aware of this issue and realize that a model can be valid without being *accurate* (Schmid, 2005) and that model's *plausibility* is substantiated when its outcomes *do not contradict previously verified theories* (Bungartz et al., 2014). Student S2 said they were satisfied with their model, "when there was a visible difference between a higher density value or a higher contagiousness value. First there was no visible difference. When there was somewhat bigger visible difference, […] we knew that it was more realistic than at first." Group G5 was unable to validate their model against real data, but observed the outcomes of their model instead and concluded their model was valid when the trends in the output data resembled what they would expect to happen in the reality. Similarly, Student S11 said, "Because you can't really with numbers — I haven't checked with numbers if it worked. But rather, what do I see, does it somewhat correspond with what I'd expect in reality."

All other groups, except G6, commented on the plausibility of their models in light of *possible contradiction with previously verified theories* or unnatural behavior. Student S3 said they were *satisfied* with their model," when, to begin with, as many people as possible left the building and it happened within realistic time frame — not like initially spending ten minutes in some corner walking to and fro." Group G3 was satisfied when their model's outcome was that the number of people in their sustainable Mars colony turned out to be relatively stable at around 17, without extreme oscillations.

On the other hand, student S7 has doubts about their model and any other model of electrons as well, "well, what is bothering me, every model is actually wrong in my view", but still goes on to say, "yet it was a good model, because, at least it showed, what we could see was that the values it produced corresponded rather well, so then we thought, we'll use it."

All the students commented on the *confidence* in their models — i.e., *credibility*, and *satisfaction that the desired results were achieved sufficiently*. Four groups found their models reasonably *credible* and were *satisfied*, albeit with the necessary, yet unspoken, reservations in the light of the context where their models were made

— that is, within computer science lessons, as their first encounter with modeling. Student S11 was *satisfied* with the model, "because the model works the way I wanted, and the results, of course I'm happy with. Because you can do something with it, say something about it." When asked about being convinced about their model, student S2 replied, "I don't think this would happen like this for real. But I wouldn't know what I'd need to improve." Student S4 commented their model, "we saw of course that the program, running once, that everyone escaped, so it was like, it worked"; then went on to say, "but I think we were happy it worked, and yeah, maybe it is a suboptimal solution" and finally concluded, "if it works and it's somewhat realistic, then we find it all right." Student S3 elucidated, "Yeah, we were happy when everyone who heard the alarm just really went outside. Not through the wall but just through a doorway." However, student S3 doubts this model is sufficiently realistic to be useful and would first need the people in the model to react to each other and to move more realistically. Student S6 commented their model, "I find that for the most part it corresponds [to reality], but if you look at the small details, we didn't find them relevant and they don't correspond in my opinion." When asked when they would consider their model a success, student S6 replied, "when people stay alive for a longer period of time, it succeeded" and went on to comment, "with the assumptions we made and the stuff we looked up, I'm quite happy with the result."

Two groups were not confident. Student S9 commented, "I know of course that this is not 100% correct, that it isn't exactly realistic. So I think to myself, can you really do something with it — that a cheese producer would really use this to determine his sale strategy, I doubt it." Student S7, when asked about being convinced about their model, replied, "Not at all. Well, it's the small things, I don't know, it doesn't work the way I was taught it works" and went on to elaborate that nobody actually knows about electrons for certain, and that the notion of an electron as accepted in modern science is also only a model.

6.5 Conclusion and Discussion

In this section we present our findings and reflect on them.

6.5.1 Findings

In answering our **first research question** — *How can the students' understanding of model verification and validation be portrayed in terms of validation techniques*

they employ? — we characterized the students' understanding in terms of these elements:

- Construct the model upon assumptions resulting from research and abstraction process.
- Test the model: generate, observe and interpret outcomes.
- Reflection after the model is built: plausibility, accuracy, credibility and satisfaction.

In answering our **second research question** — *What difficulties do the students encounter when verifying and validating their models?* — we observed a number of issues and problems:

- Performing the research necessary to build the models does not always happen. Erroneous perspectives are sometimes employed in the process of abstraction. Omissions are being made during the implementation of the model.
- When testing the model, a systematic approach to generating outcomes is seldom employed and is lacking when observing and interpreting the outcomes.
- Finally, during the reflection upon the models, there is satisfaction with a clearly unrealistic model, and even cases of not understanding the essence of modeling altogether.

6.5.2 Reflection on the Findings

Next to the validation aspects we observed, it is interesting to mention what we did not observe.

A category of validation techniques we did not observe in this study has to do with numerical aspects of modeling. None of the students reported *extreme conditions test* — making sure the models' outputs were plausible in extreme conditions (Sargent, 2013). Also, none of them used *historical data* to not only calibrate the model, but also to check whether the model behaves as the real system (Sargent, 2013). No-one performed *predictive validation*, i.e. used the model to forecast the outcomes and then compared those forecasted outcomes to the behavior of the real system (Sargent, 2013) either. Finally, no student used statistical tests or other appropriate techniques to objectively interpret the outcomes their tests. We could speculate about the reasons students did not engage in these techniques. It is plausible to think that it was difficult for them to get sufficient real data. Furthermore, their understanding of the phenomena they modeled was rather limited, which is not strange considering the position

and scope of their assignments: within a CS course, and not within a course on a particular (scientific, engineering, etc.) discipline. Finally, some of these techniques are rather advanced and belong into the repertoire of a professional modeler, rather than a student attending secondary education.

Another type of validation technique students did not report using was the engagement of external experts in any phase of their modeling process. They did not consult domain experts (Wilensky & Rand, 2015), performed no Delphi tests to seek consensus of experts on problematic outcomes (Carley, 1996), nor carried out structural walkthroughs of their models with peer groups (Sargent, 2013) — all of which they arguably could have done. None of them did a Turing test either, where they would have asked knowledgeable individuals whether it was possible to distinguish between the outcomes of the model and those of a real system (Sargent, 2013). We could have expected students to call upon experts. In one of our previous studies (Grgurina et al., 2016), we reported about our student talking to a medicine student to learn more about a particular disease; so it would not have surprised us if the students in this study had consulted their peers, teachers, or other people either to learn more about the phenomenon under scrutiny, or to seek feedback on any aspects of their models.

When we set side by side the findings from this study with the outcomes of the study we performed in 2016 (Grgurina et al., 2016), we noticed a few things.

Then, the students were expected to decide themselves what to model and some of them had difficulties coming up with a suitable problem. This time, the students choose problems from a list rather than coming up with their own problems. Consequently, all of the students this time had a clear idea of the *purpose* of the model they were developing and using.

Concerning the *research* the students did (or neglected to do), they reported similar sources of their knowledge — except this time, no-one reported consulting with an expert.

When it is time to state the assumptions the model is built upon, i.e. it is time to engage in *abstracting*, in this study we not only observed students having difficulty with this aspect of modeling — as we did in the 2016 study — but have also identified three distinct errors: *oversimplification*, *omissions* and *circular reasoning*.

Regarding the construction of the models and subsequent testing, in both of our studies, the students reported developing their models in small steps, continually testing, adjusting and expanding their models. This time we focused

specifically on verification and validation and we portrayed the students' activities and difficulties in great detail.

In this study, we asked students to extensively *reflect* on their models in terms of contentment with them, and thus we limited the scope of students' responses. To our surprise, several students declared lack of confidence in models in general — a thought we did not encounter in our previous study.

6.5.3 Reflection on the methodology

In this study, similarly to the one from 2016 (see chapter 3), a small number of students was involved which allowed us to perform an in-depth investigation of their understanding of model validation. It would be interesting to repeat the study at larger scale to see if similar practices regarding model validation can be observed.

All of the students involved in this study were in the final stages of their pre-university education, which implies that they probably experienced less difficulties that can be expected from younger students or those attending the senior general secondary education — an assertion corroborated by our findings from the study on assessment instrument (see chapter 5). Furthermore, we did not observe the students at work and only relied on the project documentation they turned in and on what they reported themselves during the interviews. Even though the researcher interviewing the students was their teacher, we have no reason to expect this fact influenced their responses.

Finally, we believe that our findings help identify both the improvements and weaknesses in the teaching materials used — as compared to teaching materials used for the previous study — and will inform the further development of the teaching materials, teaching methods and teacher training. We suggest to put more emphasis on teaching a small selection of validation techniques and giving students more guidance in using them, while simultaneously making them aware of the availability of a whole range of additional validation techniques which are not easily used within the constraints of a limited CS course in general secondary education.

Chapter 7

General Conclusions and Discussion

In this chapter, we summarize the main findings of the research project. Then we discuss its scientific contributions, reflect on the method and describe the practical implications. Finally, we present suggestions for future research.

7.1 Motivation for this Project

The integration of computational thinking into various disciplines is gaining interest in CS education where training for computational problem-solving is seen as "a truly interdisciplinary undertaking" (Tedre et al., 2018). This is exemplified in the Netherlands, where the new 2019 secondary CS curriculum introduces the topic Computational Science which is specified by the learning objectives about modeling and simulation. The curriculum describes these high-level learning objectives as follows: "Modeling: The candidate is able to model aspects of a different scientific discipline in computational terms" and "Simulation: The candidate is able to construct models and simulations, and use these for the research of phenomena in that other science field." Additionally, modeling itself is a part of the compulsory core curriculum, described as "Modeling: The candidate is able to use context to analyze a relevant problem, limit this to a manageable problem, translate this into a model, generate and interpret model results, and test and assess the model. The candidate is able to use consistent reasoning." (Barendsen & Tolboom, 2016). With the introduction of Computational Science into the CS classroom, the need arises for validated guidelines to enable and facilitate its teaching and learning. This practical need instigated our research project and we translated it into scientific research questions in order to contribute to the research knowledge on Computational Science.

This chapter is organized as follows: Section 7.2 presents our aim and research questions and provides their answers, and then discusses them and describes their scientific contribution in the light of existing literature. Section 7.3 reflects on the method. Section 7.4 reports practical implications and Section 7.5 provides suggestions for further research.

7.2 Conclusions and Discussion

So far, research into teaching modeling and simulation[24] as generic scientific competences within a CS course has been an underexposed issue in the computer science education research (CSER). There was no operational definition of the learning objectives modeling and situation, little was known about suitable

24 In this thesis, we use terms *modeling, modeling & simulation* and *Computational Science* interchangeably, unless explicitly stated otherwise.

teaching and assessment strategies and students' understanding; and teachers' ideas about teaching modeling and simulation have not been charted either. Therefore, the aim of our research project is to explore the pedagogical aspects of teaching Computational Science in the Computer Science course in secondary education.

We looked both at the pedagogical aspects of teaching of Computational Science (i.e., modeling and simulation) and at teachers' practical knowledge about these pedagogical aspects through the lens of Magnusson's (1999) components of topic-specific pedagogy — that is, in terms of:

- M1 goals and objectives
- M2 students' understanding including requirements for learning and their difficulties
- M3 instructional strategies
- M4 methods of assessment.

We translated this global aim of our research project into these four research questions:

- RQ1 What computational thinking activities constitute the problem-solving process associated with Computational Science? This question aims to find an operational definition of the learning goals and objectives of Computational Science. (M1)
- RQ2 How can the students' understanding of modeling activities be portrayed in terms of their requirements for learning and difficulties they encounter? (M2)
- RQ3 What are characteristics of a valid and reliable assessment instrument for Computational Science? (M4)
- RQ4 How can the teachers' Pedagogical Content Knowledge (PCK) for teaching Computational Science be portrayed in terms of the four components M1 to M4?

7.2.1 Operational Definition of Computational Science (RQ1)

The first empirical study (chapter 3) focused on obtaining an operational definition of the learning objectives modeling and simulation — Magnusson's component M1 — to answer our first research question: What computational thinking activities constitute the problem-solving process associated with Computational Science (i.e., modeling and simulation)?

We characterized modeling and simulation from two perspectives: first, as a cyclic process with distinct stages derived from the mathematical modeling process and simulation modeling, and second, as a means to integrate computational thinking and science from the perspective of CS students who develop computational models and use them for scientific enquiry.

We have obtained an operationalization of the learning objectives for modeling and simulation in terms of an iterative process framework consisting of the following elements:

- *stating* the purpose of a model
- engaging in the *research* necessary to build the model
- performing *abstraction* to take into account only the relevant aspects of the phenomenon under scrutiny
- *formulating* the problem in a way that allows the use a computer and other tools to help solve it
- stating the *requirements* and *specification*
- *implementing* the model, i.e., programming it
- performing *verification* and *validation* of the implemented model
- using the implemented model to perform the *experiment*
- *analyzing* the data obtained from the experiment
- *reflecting* on the whole process.

This operational description of the learning objectives of Computational Science provides a framework for the engagement in scientific practices through the development and use of computational models. It brings modeling and simulation within reach of secondary students: it characterizes modeling as a cyclic process analogous to a mathematical modeling process, yet contrary to it, the models produced are *executable*. In this sense it differs from existing operationalizations of modelling in scientific literature, which focus on the development of *static* models. So far, the research on modeling focused mostly on mathematics and there are common aspects recognized in mathematics as well. In mathematics, the development and use of models is also considered to be a cyclic process with distinguished stages: definition, conceptualization, formalization, execution and conclusion, and additionally, reflection (Overveld et al., 2015; Perrenet & Zwaneveld, 2012). Formalization and execution stages are considered to be a part of mathematical world and involve mathematical formalizations such as formulas and (differential) equations. In simulation

modeling, on the other hand, these two stages are interpreted in computational terms. Formalization means constructing a computational model in the form of a computer program. That is in itself a cyclic process which cycles through the phases of establishing requirements and specification, designing the program, and its testing and evaluation. This cyclic formalization process is thus embedded within the encompassing modeling cycle. This aspect, too, signifies that modeling in Computational Science is essentially different from modeling in mathematics. The execution aspect entails designing and running experiments, thus using the computational model to perform simulation (Law, 2015). This framework is a novelty as it provides a detailed operationalization of the learning objectives for modeling and simulation within CS education.

The scientific value of our framework lies in the fact that it can serve as a conceptual model for the investigation of students' cognitive activities in empirical studies, i.e., to explore their understanding and difficulties while engaging in modeling and simulation tasks through the development and use of computational models. This methodological application is particularly important since the application of computational thinking in modern science education is gaining interest. According to Lee et al., (2020), computational thinking "is seen as having the potential to deepen STEM[25] learning by positioning students as young scientists and innovators through engagement in authentic STEM practices". Regarding modeling itself, Gilbert & Justi (2016) state that it plays a significant role in the development and learning of science (Gilbert & Justi, 2016), as do Hallström & Schönborn (2019) who believe that it contributes to authentic STEM education. Similar to our framework, but with less detail, Sengupta et al. (2013) propose a theoretical framework for integrating computational thinking with science in primary and secondary education. In that framework, *learning-by-doing* activities are also represented as a cyclic process where students iterate between scientific enquiry (i.e., understanding of the scientific phenomena and modeling practices), algorithm design (i.e., development of a computational model) and engineering (i.e., refining models and simulations). Our framework is therefore particularly useful for research of computational thinking in context where computational modeling is used to enhance STEM learning. Furthermore, our framework provides an interpretation of modeling and simulation within CS which is not geared toward the use a specific software tool. In that sense, it is new as it shifts focus from the production of computational artifacts to embedding

25 Science, Engineering, Technology and Mathematics

computing in other disciplines with the goal of helping to solve problems in these other disciplines. After we have finished this work, Sengupta et al. (2018) confirmed our ideas in their recommendations where they warned against a technocentric focus on production and use of computational artifacts and argued "that computational thinking must be reconceptualized more appropriately as an intersubjective experience" — exactly as we did.

7.2.2 Students' Understanding and Difficulties (RQ2)

In our first and fourth studies, we focused on students' understanding and difficulties while engaging in modeling activities — Magnusson's component M2 — and sought to answer our second research question: How can the students' understanding of modeling activities be portrayed in terms of their requirements for learning and difficulties they encounter? In the first study (chapter 3), we looked at the specific challenges the students experience when engaging in modeling activities in all of the modeling process. In the fourth study (chapter 6) we focused on students' challenges related to the verification and validation aspect of the modeling process only.

Our first study resulted in the qualifications of the challenges the students faced when developing and using computational models. Our in-depth analyses revealed that students face two types of challenges related to the two cyclic processes contained in our framework: those related to the entire modeling cycle and those related to formalization — i.e., the development of a computer program — which is an element of the modeling cycle. The challenges which we found that were related to the entire modeling cycle are those involving the context of the discipline where the problem at hand originates. They are related to expressing the problem at hand in computational terms, interpreting the computational solution in terms of the original subject matter, and reflecting upon the whole process. The difficulties our students faced while constructing their models were also reported in case of students constructing mathematical models: wrong level of abstraction and erroneous assumptions (Maaß, 2006). Not being familiar with the affordances of the tool used to implement the model — in our case, insufficient command of the programming language involved — is found to have a detrimental effect on the quality of the model being produced (Bielik et al., 2021; Sins et al., 2005). Other behaviors we identified are characteristic for the development and use of computational models: not knowing whether unexpected behavior of a model is caused by an error or emergent behavior is typical for the development of agent-

based models, as is incremental model development (Wilensky & Rand, 2015). Finally, the two cycles constituting the modeling process cause confusion in students when they do not know to which cycle to attribute a particular occurrence. Our framework provides a refinement of existing frameworks to characterize and investigate this confusion in students.

The challenges which we found that were related to formalization are typical for computer science and characteristic for the construction of a computational model, thus they are concerned with programming, testing, and debugging a computer program. Our students were not novice programmers and they reported taking care of the related problems themselves. We discuss this finding in terms of two aspects: first, in terms of the construction of a computer program, and second, in terms of correctness of a computer program. Regarding the construction of computer programs, Qian & Lehman (2017) examined flawed or incomplete understandings of learners of introductory computer programming through the framework consisting of three elements. First element is syntactic knowledge, i.e., knowledge about the language features, basic rules and facts, such as for example use of semicolons. Second element is conceptual knowledge which is concerned with the programming constructs and inner workings of a computer. Third element is strategic knowledge which is concerned with the application "of syntactic and conceptual knowledge of programming to solve novel problems". Taken together, these three elements describe a student's ability to construct a working program. Our research adds the perspective of students with more programming experience. Our students were not novice programmers since they already had programmed in Python. Even though both constructing models and programming them in NetLogo were new to them, they reported taking care of their programming problems themselves. In other words, they all managed to develop working programs, i.e., to construct working models. This finding illustrates that they were able to use their programming skills in a new context, where they successfully constructed models. Regarding the correctness of the programs, Kolikant (2005) found that students rarely engage in systematic testing and debugging of their programs and have been found to consider a program to be correct even when it demonstrates incorrect behavior. In our specific situation, the formalization step meant that the students had an open programming assignment where we did not provide input and output values to test their programs against. This allows a possibility that our students accepted incorrect programs — which seemingly worked properly — as being correct. We have accepted these programs

as correct too, and have not performed additional testing or formal verification to examine their correctness as doing so was beyond the scope of this research project.

In our final study (chapter 6), we explored students' understanding and difficulties while working on Computational Science assignments using the teaching materials we developed ourselves. We focused on their understanding and difficulties concerning verification and validation of the models they develop. We characterized their understanding in terms of these elements:

- Construct the model upon assumptions resulting from research and abstraction process.
- Test the model: generate, observe and interpret outcomes.
- Reflection after the model is built: plausibility, accuracy, credibility and satisfaction.

Looking at the difficulties the students encountered when verifying and validating their models, we found that:

- Not all students explicitly engaged in research necessary to build their models. In the process of abstraction, some students employed erroneous perspectives. During the implementation of the model, the students reported making omissions.
- When testing the model, a systematic approach to generating outcomes was employed only by some of the groups and was lacking when observing and interpreting the outcomes.
- When reflecting upon their models, some students were satisfied with a clearly unrealistic model, and some appeared not to understand the essence of modeling altogether.

To discuss our findings, we note that students' difficulties with validation of their models have — to our best knowledge — barely been explored in the context of computer science education. When Louca et al. (2011) asked their students to construct computational models, they only assessed the surface structure of the implemented models, and only labeled them as correct or incorrect. In mathematics education, difficulties with validation of models were explored more extensively (Eraslan & Kant, 2015), as was satisfaction with unrealistic models that was reported in students engaging in mathematical modeling too (Edo et al., 2013; Maaß, 2006). The scientific value of our approach stems from the distinguishing characteristic of our studies: they took place in a CS class where

technical aspects of developing and using models were of prime concern, while the use of models for scientific inquiry remained limited to providing a context where the models were developed. In that sense, our approach differs from the one described in many existing studies. These studies described students constructing (computational) models where the disciplinary content of the application domain was of principal interest. The (computational) aspects of the model construction played a secondary, supportive role as a means to reach that goal — cf. Basu et al. (2016), Bielik et al. (2021), Eraslan & Kant (2015), Maaß (2006) and Sins et al., (2005). Contrary to that approach, we focus specifically on the embedding of computational modeling into application domain from the point of view of computer science. The approach of Sins et al., (2005) is illustrative for this difference. In their study, in the context of a physics course, the students were given a partial computational model and empirical data for a particular phenomenon, and were then asked to finish that model. In our studies, on the contrary, students were given open questions and asked to answer them through the construction and use of computation models through the steps defined in our framework. To illustrate this point, we draw on an example from our third study which focused on the assessment instrument (see chapter 5). Several student groups were answering the question whether sustainable human life was possible on Mars. While the models produced differed greatly, however, all of the students' answers could possibly have been considered to be correct despite their variations. Indeed, since our approach emphasizes the technical aspects of developing and using models, our findings are independent of any specific application domain.

7.2.3 Assessment (RQ3)

In our third study (chapter 5), we focused on assessment — Magnusson's component M4 — in order to answer our third research question: What are characteristics of a valid and reliable assessment instrument for Computational Science?

In that study, we focused on the assessment instrument which we developed along with our teaching materials. The instrument has the form of a practical assignment with the accompanying grading rubrics. The assignment follows closely our framework for the engagement in scientific practices through the development and use of computational models. It contains a series of questions and tasks which guide the students through the whole process: to explain the purpose of the model (i.e., to state the research question) and to perform any

necessary research; to design, implement and validate the model; to perform experiments by executing the model, to analyze the outcomes and to answer the research question; and finally, to reflect on the entire process. This results is a portfolio containing documentation and the implemented computational model. The grading rubrics classifies the learning outcomes for each part of the portfolio using the Structure of the Observed Learning Outcome (SOLO) taxonomy which describes the learning progress through five levels of understanding: prestructural, unistructural and multistructural — which are considered to be quantitative — and relational and extended abstract — which indicate a qualitative change (Biggs & Tang, 2011). Our assessment instrument in the form of a practical assignment and accompanying rubrics based on the SOLO taxonomy proved to be reliable, as indicated by a high rate of inter-rater agreement. Its validity is corroborated by exposing the significant differences in the performance levels of the HAVO[26] students compared to the VWO[27] students: as expected, the performance levels of the VWO students were significantly higher for almost all the criteria.

So far, assessing computational thinking has received a lot of interest, as reported in mapping and review studies by de Araujo et al. (2016), Martins Pacheco et al. (2019) and Tang et al. (2020). A typical example is provided by Roman-Gonzalez et al. (2017) who present their Computational Thinking Test. This multiple-choice test assesses students' knowledge of computational concepts and is as such focused on programming, independent of any specific context. A number of other examples of assessing computational thinking focus specifically on modeling.

To compare our results to these other studies, we discuss our results across three dimensions: the context where assessment instrument is used, the quality of the rubrics used and the aspects of the modeling cycle involved. Inevitably, some of the discussion will touch upon the nature of the modeling process itself, since assessment is inseparable from it.

Teaching computational modeling is often situated in a specific *context* where attention is given to the learning objectives related both to the subject matter and to the computational aspects. Consequently, both of these learning objectives are assessed. For example, Caballero et al. (2012) described students who developed computational models of the motion of a craft orbiting Earth by completing a partially completed program. Incorrect programs were analyzed to unveil

26 HAVO: in Dutch: hoger algemeen voorbereidend onderwijs: senior secondary education
27 VWO: in Dutch: voorbereidend wetenschappelijk onderwijs: pre-university education

students' difficulties related to the algorithmic approach and to count the errors related to each of the three procedural areas related to common student mistakes. Similarly, Basu et al. (2018) developed assessment tasks for the integration of CT in physics and again, the students were asked to complete a partially completed program. The *rubrics* used for the assessment assessed two aspects: expressing physics relations in a computational model and using programming concepts to model physics phenomena. In both of these examples, physics provides the modeling context. The aspects under scrutiny were labeled either as correct or incorrect. As to the modeling cycle, in comparison to our assessment instrument, only the design and the implementation of the models were of interest, rather that the whole *modeling cycle*. Louca et al. (2011) analyzed the computational models of a number of physics phenomena constructed by their students by using categories for the representation of objects, entities, behaviors and interaction, and additionally, for the accuracy of the phenomenon description. Each of these categories contains a number of subcategories specific for the context. Taken together, these categories are reminiscent of the description of the attainment levels specified in our rubrics. In further comparison to our assessment instrument, we see that in addition to design and the implementation of the models, their validity is of interest too — albeit to a limited extent, as discussed in the section 7.2.2 on students understanding and difficulties.

Other researchers have constructed assessment for models in their own right. For science teaching and learning, Papaevripidou et al. (2014) describe modeling competence in terms of modeling practices and meta-knowledge. They identify four modeling practices: construction, use, comparison and revision of models; they distinguish different levels of increasing sophistication for each of the modeling practices and they observe that these levels are independent of the modeling tool. While they are not concerned with computational models, we see similarities in the modeling practices they mention: construction and use are represented in our modeling cycle too, and their revision of models is represented in our framework in the fact that our framework considers modeling to be a cyclic process. Furthermore, for each of the modeling practices, they distinguish several levels of sophistication — an approach seemingly similar to our five ordered categories of SOLO taxonomy. However, they assess each of these practices as a whole. For example, for the practice of model use, they assess efficient use of the model without specifying details that characterize efficient use, while we provide detailed characterization of each of the levels in our assessment instrument.

In reflecting on our results, we emphasize the three aspects where our assessment instrument for Computational Science differs from the existing assessment instruments. First, it situates computational modeling in the context of CS education, independent of any domain specific context. Second, it covers the whole of the modeling cycle described in our framework, but we do remark that it does not deeply scrutinize the program code, as opposed to many other assessment instruments related to computational modeling. Third, our instrument uses a five-level rubrics based on SOLO taxonomy (Biggs & Tang, 2011) to assess the element of our modeling cycle framework.

We note that our assessment instrument aligns well with the suggestions regarding the assessment of CT which were put forward by Tang et al. (2020) after we have finished this research project. We go on to discuss them in detail and observe that our work adheres to most of these suggestions. We contributed to creating more assessment for high school (as opposed to elementary and middle school). Our assessment focuses on the integration of CT and subject matter by focusing on computational modeling and simulation to be used in a different discipline in the context of scientific enquiry. We report the validity and reliability of the assessment. We view CT broader than programming or computing only. To a high degree, we designed "CT assessments that can be applicable across platforms and devices". Namely, even though we have developed our assessment instrument to be used in the context of scientific enquiry when constructing and using agent-based models, with slight modifications it could be used with other computational models as well. Our assessment does adhere to the rest of this suggestion: "in order to compare students' CT performance under varied conditions of intervention". Finally, when we look at our assessment as an instrument to be used by teachers in their daily teaching practice, we note that it does not follow their suggestion "to consider the concurrent use of qualitative measures collected by interviews, think-alouds, or focus groups to better understand students' proficiency of CT". In our first study (chapter 3), we scrutinized a number of qualitative measures for visible occurrences of the elements of our modeling framework. While our findings confirm that interviews with students (as well as close observations of student groups during their work) provide rich insights into students' performance, understanding and difficulties — and therefore serve well as research instrument — we chose not to include them into our assessment instrument because they are not feasible in everyday teaching practice. Additionally, we point out that our assessment instrument also aligns well

with the recommendations for the assessment of modeling competence as such, put forward by Nicolaou & Constantinou (2014), when they suggest to formulate rubrics of students attainment level with regards to modeling competences.

The scientific value of our assessment instrument lies in the fact that is useful beyond the classroom — it can be employed as a *research* instrument when examining the students' learning outcomes with respect to computational thinking. Furthermore, the attention our assessment instrument puts on the entire modeling cycle makes it more holistic than most of the other assessment instruments with narrower focus. Finally, next to other forms of validity, we explored one which is essential to the Dutch educational context — the discernment ability to expose the significant differences in the performance levels of the HAVO students compared to the VWO students.

7.2.4 Teachers' PCK (RQ4)

In the second study (chapter 4), we portrayed computer science teachers' initial pedagogical content knowledge (PCK) on modeling and simulation in order to answer our fourth research question: How can the teachers' PCK of teaching Computational Science be portrayed in terms of the four elements of PCK? Additionally, we asked, What differential features of PCK can be used to identify patterns of individual PCK in terms of the four elements of PCK?

First, we characterized the teachers' PCK and portrayed it in terms of the four components of PCK.

Concerning teachers' knowledge about goals of objectives (M1) on teaching modeling, we found two types of learning objectives. First, there are conceptual objectives concerning the skills associated with CS subject matter, and second, there are motivational and practical objectives concerning transversal competences and understanding the benefits of models.

Concerning the teachers' knowledge about students' understanding (M2), we characterized it in terms of three issues. First, the prerequisite knowledge the students need to learn modeling and skills needed to make models. Second, the issues regarded as successful or contributing to success such as the relevance of the models, the students' perception, technical aspect and interest. Third, the issues regarded as difficult or contributing to difficulties, such as variation among students in the class, students' difficulties in understanding the nature of models, or with abstraction or formalization, and students' approach to task at hand. Furthermore, we observed that some teachers do not know what to say about the successful or difficult issues.

Concerning the knowledge about instructional strategies (M3), we described it in terms of five issues: the perceived role as teachers, assignments to be given to students, student's characteristic to be taken into account, organizational aspects, and finally, difficulties and problems. We observed an agreement about subject-specific strategies — scaffolding learning with a final project which serves both to give students the opportunity to learn how to develop a model from scratch and as assessment.

Concerning the knowledge about assessment (M4), we described it in terms of four issues: the form of the assignment, problems given to students to work on, organizational issues, and finally, the assessment criteria. We observed an agreement about a suitable assessment form — a large practical assignment. Furthermore, we found great variation in the knowledge of dimension to assess, and in granularity and depth of the description of assessment criteria.

Additionally, we described two characteristics that distinguish among teachers — their focus on conceptual versus motivational and practical learning goals and objectives (M1) and their emphasis on product-based versus process-based assessment (M4) — leading to four distinct groups of teachers. However, none of these differential features leads to an overall typification of the teachers' PCK.

The construct of pedagogical content knowledge (PCK) has proven to be a powerful one to help capture teachers' views and knowledge on teaching various topics, for example in science (Henze et al., 2008), mathematics (Baumert et al., 2010) or design of digital artifacts (Rahimi et al., 2016). It is finding its way into the CS teaching as well. In a review of research literature, Hubbard (2018) reports 19 studies concerned with PCK in computing education specifically concerned with teaching computing as its own subject, and these studies are mostly concerned with programming, cf. Saeli (2012). A number of other studies which are concerned with teachers' PCK of computer science or computational thinking focus on programming as well (Yadav et al., 2016; Yadav & Berges, 2019) or specifically on programming in the context of robotics (Çakıroğlu & Kiliç, 2020; Chalmers, 2018). Our study seems to be unique with its focus on the PCK of modeling and simulation in the context of CS education. Yet, we can compare the findings about our teachers' PCK to the results of our studies on students' understanding and draw parallels with other studies of teachers' PCK related to CS or modeling.

Our studies on students' understanding confirmed that students indeed faced difficulties related to understanding the nature of models and found abstraction

and formalization challenging, exactly as reported by teachers. Furthermore, teachers were right about the inefficient students' strategies. However, where teachers in our study mentioned no misconceptions, we did find them, for example, related to the nature of models.

We now go on to reflect on the method used for this study. Considering the explorative character of our study, we felt that a qualitative research method was appropriate to capture the whole breadth of teachers' opinions and ideas. Therefore, we conducted semi-structured interviews with our teachers as did, for example, Liberman et al., (2012) and Griffin (2016), rather than, for example, present teachers with teaching vignettes with close-ended responses and focus on students' understanding only (cf. Yadav & Berges (2019)). Our research yields portraits of CS teachers' PCK and, unlike Saeli (2012), we refrain from *assessing* their PCK.

Rahimi et al. (2016), performed a study with similar methodology when they explored PCK of Dutch CS teachers regarding the design of digital artifacts. Their findings describe teachers' PCK in terms similar to the ones we found. However, unlike them, we were not able to typify teachers' PCK through relating their knowledge of students' understanding and instructional strategies on one hand, to their knowledge of goals and objectives and knowledge of assessment on the other. Henze et al. (2007) explored science teachers PCK on models and modeling, also in the context of the Dutch secondary education. They, too, were able to distinguish two types of teacher knowledge. If we speculate about the reasons why, in our case, we were not able to identify specific types of teacher knowledge, we should consider the novelty of modeling and simulation in the context of CS education, and teachers' lack of experience in teaching it. So, while Rahimi et al. (2016) saw that certain components of PCK were predictive of other components, in our case there is a lot of variation and not much consistency among various components of teachers' PCK which suggests the reasons why we could not establish such typification.

7.2.5 Overall Contribution

Computer science, teaching CS and research into the teaching of CS are rather young disciplines, especially when compared to, for example, mathematics. As we indicated in the introduction of this thesis (see chapter 1), computational thinking — and its component modeling and simulation — can form a bridge between CS and an application domain. A lot of research in CS education is dedicated to

programming education and novice programmers. However, we are not aware of research about employing programming skills to develop computational models which are used for scientific enquiry in the context of CS education. It is in this light that we see the scientific contribution of this project to the development of the theory of the CS education. Our work provides a novel theoretical framework for empirical research into computational modeling competences, principles for their assessment and insights into students' understanding regarding these competences. Furthermore, it is the first instance where the related teachers' PCK is explored.

7.3 Reflection on Methodology

In this section, we reflect on the methods applied in our research project.

To explore the pedagogical aspects of teaching Computational Science, we employed the lens of Magnusson's (1999) components of topic-specific pedagogy. This approach allowed us to organize and structure our research project and investigate pedagogy in a feasible manner. Furthermore, our project provides evidence that Magnusson's view of components of topic-specific pedagogy can successfully be applied in the context of CS education as well.

We carried out this research project with a limited number of participants. The participating teachers were from the local CS teachers' network who replied to our invitation to be interviewed and were in that sense self-selected. However, we believe that this fact did not influence our findings significantly because this sample was still reasonably representative for the whole of the CS teacher population in the Netherlands regarding teachers' own educational background, teaching qualifications, and experience with teaching (as described in chapter 2). The same holds true for the students from the two schools participating in the project. As a consequence of the centralized and regulated way the Dutch education is set up — especially in the upper grades of HAVO and VWO — the cognitive abilities of all of the students attending these types of school are expected to be fairly uniform (see chapter 2 for a description of Dutch educational system). We believe that working with a small sample and familiarity with the teaching circumstances provided us with better understanding of what was going on in the classroom and made it possible to perform in-depth qualitative analysis of the teachers' ideas on teaching and of students' learning.

The studies in this research project were carried out using the pilot version of teaching materials developed by the author of this thesis as the first step of our educational design research (Akker et al., 2006). Later, as a spin-off of this project, new teaching materials were developed by a team of experienced CS teachers lead by her, and they took into account the findings from this research project, as described in section 7.4 on practical implications of this project. However, we do not expect that using the pilot version of the teaching materials caused our results to be less reliable because the essential elements were present in that pilot version. Therefore, we have a reason to believe that the results would have been the same if we had used the new teaching materials instead.

7.4 Practical Implications

In this section, we reflect on the practical implications of our research project. We first describe the scientific paradigm shifts which made it possible to bring Computational Science into a secondary classroom, and then go on to discuss the practical implications.

The overarching practical contribution of this research project can be seen clearly by considering the shift of science paradigms (Hey et al., 2009) brought about by restructurations, i.e., "reformulating knowledge disciplines through new representational forms" (Wilensky & Papert, 2010). Many great theoretical scientific achievements — such as classical mechanics or Lotka-Volterra equations describing the dynamics of biological systems — were made possible through the developments in mathematics (which were, in turn, often driven by scientists' needs). The corresponding restructuration meant that the scientific knowledge could be described in terms of mathematical terms rather than as narratives, bringing about a science paradigm shift from the description of observations to the use of mathematical models. The development and use of such mathematical representations is often a complex process. Additionally, advanced mathematical knowledge of calculus is necessary, so an active engagement in scientific activities in this manner is often beyond reach of secondary students. However, nowadays, the advances in possibilities offered by modern computing make it possible to describe phenomena in computational terms by simply describing the characteristics and behavior of the individuals forming the system which is being modeled. These computational descriptions — i.e., computational models

— are executable and in the spirit of yet another science paradigm shift: the attention is moving from the theoretical approach to the computational approach which focuses on the simulation of complex phenomena through the use of computational models that are executable — rather than on modeling them only. That means that nowadays even novices — such as secondary students — have tools at their disposal to actively engage in scientific practices. Indeed, Weintrop et al. (2016) observe that science is increasingly becoming a computational endeavor and they consider modeling and simulation practices to be one of the main categories of computational thinking for mathematics and science. Our project provides evidence that it is possible to teach modeling and simulation effectively within a secondary CS course and to empower students to embark on a journey of *doing science* themselves.

In chapter 2, we described the history and situation of secondary CS education in the Netherlands. It is in this light that we see the principal practical contribution of this research project: it informs the teaching of Computational Science. Our results contributed to the design of professional development activities for in-service and pre-service teachers, covering both the aspects of modeling and simulation, as well as the pedagogy suitable to teach it successfully. Furthermore, as a spin-off of this project, we used our findings to guide the development of teaching materials and accompanying teachers' manual which are now a part of CS textbooks[28] available to all secondary students in the Netherlands. This way, we address a number of critical factors listed in the report about the state of secondary CS education in the Netherlands (Tolboom et al., 2014) (see section 2.3.2.2): we created modular teaching material in order to provide for rapid advances of the discipline, we contribute to the in-service training of the teachers, and with our assessment instrument, we contribute to the quality of assessment in schools.

In a broader perspective, our findings could influence teaching of other CS content. They could alert teachers about understanding and difficulties they could expect from their students, and inspire the development of a holistic assessment instrument like ours.

When we embarked on this project, we focused on modeling and simulation within CS education. We hoped it would contribute to building bridges to other school courses and compel our students and their teachers to reach out from the confines of their disciplines. We hoped to effectuate interdisciplinary cooperation which would benefit students' learning of all the disciplines involved. We are glad

28 https://ieni.github.io/inf2019/themas/r-computational-science

to observe that since then, computational thinking has been spreading steadily, and that our work is gaining relevancy. If we consider the three steps constituting the CT problem-solving process — expressing the problem in computational terms, constructing a computational solution, and interpreting that computational solution in terms of the original subject matter (Barendsen & Bruggink, 2019) — we see that computational thinking is increasingly considered to play a central role in science education (Park & Green, 2019), and computational models in particular (Pears et al., 2019).

When we look outside the CS education, we observe that effectively using and understanding computational models does not necessarily require students to develop them from scratch themselves. Our work is shown to be relevant outside of the CS education as our findings already inform the design of instructional strategies for younger students where models are used to enrich the teaching of other subjects — as for example in the TeaEdu4CT[29] project concerned with teacher education for CT and STEAM[30]. When computational thinking is integrated into the context of another discipline, students can use an existing computational model (for example about the spread of a virus) to run simulations, analyze outcomes and examine consequences — both scientific and societal. Additionally, they could be stimulated to discuss the assumptions underlaying the model and the model's validity, thus effectively engaging in a number of processes associated with the modeling cycle and *doing science*.

This example is indicative of the curriculum changes expected to take place in the Netherlands, as discussed in chapter 2. Today, the stakeholders recognize the importance of learning computer science, as demonstrated by the curriculum.nu[31] initiative where teachers and school administrators cooperate to modernize the curriculum for elementary and lower secondary education. This initiative intends to introduce a new learning domain Digital Literacy which contains four elements: ICT skills, Media Wisdom, Computational Thinking and Information Skills (Thijs et al., 2014). The Computational Thinking element of Digital literacy covers a number of CS specific topics, including programming. If the proposals put forward by the curriculum.nu initiative get approved and lead to the introduction of a new curriculum for elementary and lower secondary education, it would signify the end of a rather unique situation where the Netherlands found itself in comparison to many modern nations. As opposed to, for example, England

29 https://cesie.org/en/project/teaedu4ct/
30 STEAM: science, technology, engineering, art and mathematics
31 https://www.curriculum.nu

(Barendsen et al., 2015), Denmark (Caspersen & Nowack, 2013) or Lithuania (Dagienė & Stupuriene, 2016), today in the Netherlands there is no compulsory CS education at all for students in primary education. In secondary education, the Dutch students get the opportunity to enjoy CS education only if they attend specialized vocational schools (in Dutch: VMBO[32]); otherwise, the students attending senior secondary education (in Dutch: HAVO) or pre-university education (in Dutch: VWO) have to wait until the 10th grade to attend the elective CS course — and only if their school choses to offer this course and is able to find a CS teacher.

Considering the present situation of the CS education in the Dutch primary and secondary education — the actual situation at the moment — we can only conclude that it is precarious. We urge those at the helm — the policy makers — to take the right decision and bring Digital literacy into all of Dutch primary and secondary education. That way, all the students involved will finally get a taste of computer science.

7.5 Suggestions for Further Research

This research project took place largely before and partially in parallel with the development of the new 2019 secondary CS curriculum in the Netherlands. While its results informed the development of the curriculum — in particular the elective theme Computational Science — the effects and results of the curriculum implementation are not yet examined. Looking broader, we see an explosion of efforts to teach computational thinking — a notion for which Computational Science is a prime exemplar. We therefore present a number of suggestions for further research.

The study on the teachers' initial Pedagogical Content Knowledge (PCK) regarding Computational Science was carried out before the CS curriculum containing this elective theme came into effect, with a small number of teachers, and we portrayed their PCK at that moment. We suggest to perform a large in-depth study of the development of teachers' PCK regarding Computational Science as they participate in relevant professional development activities and teach it, with the aim to explore how their PCK evolves as they gain experience teaching Computational Science.

32 VMBO: in Dutch: voorbereidend middelbaar beroepsonderwijs: prevocational education

We developed teaching materials to support the research of students' understanding (M2) and methods of assessment (M4). However, the instructional strategies (M3) themselves were not explored. We suggest to perform research about successful instructional strategies for Computational Science. The teaching materials developed as spin-off of this project and our assessment instrument can form the basis for this research.

Our assessment instrument based on the SOLO taxonomy is in line with the suggestions and needs expressed by CS teachers and provides for holistic assessment of the learning objectives related to Computational Science. We suggest to research the development of similar assessments instruments focusing not only on the computational concepts, but also computational practices and computational perspectives for, on one hand, other learning objectives of computer sciences, and on the other hand, for learning objectives within other disciplines where computational thinking is involved.

In this project, we focused on research of teaching Computational Science from within a CS course. Computational Science aims to provide CS students with tools, techniques and skills to use modeling and simulation when exploring phenomena in various scientific disciplines outside of CS. We suggest to extend this research along three dimensions. First, following our original line of enquiry further and considering that computational models can produce large quantities of data, we suggested a line of inquiry in the wake of another science paradigm shift — from computational approach that models and simulates complex phenomena to the one that focuses on the exploration of data and unifies theory, experiment and simulation (Hey et al., 2009). By engaging in the practices of data science that bring together computational thinking and mathematical thinking, the students developing models and performing simulations with them would be given a further opportunity to engage in *doing scienc*e by means of more thorough analysis of the data produced by their simulations. Since this type of activity happened only marginally within this project, we suggest further research into this specific issue. Second, extending the scope, we propose a new vantage point — the perspective of computational thinking in context — and suggest to explore students' understanding, challenges, and difficulties related to the learning of the disciplinary content for which they make models and perform simulations as described in the learning objectives of Computational Science. Third, as an extension of the previous suggestion, we propose to explore pedagogical aspects of

teaching digital literacy, and in particular its component computational thinking, if and when they become a part of the curriculum for the primary and lower secondary education in the Netherlands.

Chapter 8

Nederlandse samenvatting Modelleren en simuleren binnen Informatica in het voortgezet onderwijs

8.1 Motivatie en Onderzoeksvragen

Informaticaonderwijs is ontstaan in het kielzog van de opkomst van computers in de jaren '50 van de vorige eeuw. De doelen van het Informaticaonderwijs zijn meegeëvolueerd met de toepassing van computers: eerst gericht op het opleiden voor technische banen, en later — met een ruimere beschikbaarheid van computers in de tweede helft van de 20ste eeuw — voor softwareontwikkeling en academisch onderzoek. Vandaag de dag, met computers die in allerlei vormen en maten elk aspect van ons professionele, sociale en privéleven doordringen, wordt Informatica niet alleen onderwezen in het kader van voorbereiding op de arbeidsmarkt, maar ook om computational thinking[33] (CT) en digitale geletterdheid te ondersteunen, om gelijke kansen te bevorderen en om burgerschap, wetenschappelijke, technologische en maatschappelijke innovatie, onderwijsvernieuwingen en -hervormingen, en tenslotte, plezier, voldoening en persoonlijke bekwaamheid te stimuleren.

In Nederland is in 1998 Informatica als middelbare schoolvak ingevoerd als een keuzevak in de bovenbouw van HAVO en VWO. Het examenprogramma van dit vak werd herzien in 2007, en recentelijk weer vernieuwd. Vanaf het schooljaar 2019/2020 is dit nieuwe examenprogramma ingevoerd. Eén van de keuzethema's in dit nieuwe examenprogramma is Computational Science[34], dat uit twee eindtermen bestaat: modelleren en simuleren. Deze eindtermen worden als volgt omschreven: "Modelleren: De kandidaat kan aspecten van een andere wetenschappelijke discipline modelleren in computationele termen. Simuleren: De kandidaat kan modellen en simulaties construeren en gebruiken voor het onderzoeken van verschijnselen in die andere wetenschap." Daarnaast is modelleren als een onderdeel van de verplichte kerndomeinen in het vak Informatica als een wetenschappelijke vaardigheid opgenomen en beschreven als volgt: "De kandidaat kan in contexten een relevant probleem analyseren, inperken tot een hanteerbaar probleem, vertalen naar een model, modeluitkomsten genereren en interpreteren, en het model toetsen en beoordelen. De kandidaat maakt daarbij gebruik van consistente redeneringen." (Barendsen & Tolboom, 2016).

33 In de Nederlandse samenvatting blijft de Engelse begrip computational thinking gehandhaafd wegens duidelijkheid en consistentie.

34 De begrippen Computational Science, modelleren en simuleren kunnen in deze tekst als synoniemen gezien worden.

In bredere zin wordt modelleren als een integraal onderdeel gezien van Computational Thinking (CT), een concept dat Wing in 2006 breed onder de aandacht bracht. CT omvat een reeks vaardigheden waarmee men problemen kan oplossen met behulp van concepten en procedures uit de Informatica. Het gaat dus om het proces van probleem oplossen waarbij eerst het probleem in computationele elementen wordt vertaald, vervolgens een computationele oplossing wordt geconstrueerd, (bijvoorbeeld door gebruik van een bestaande applicatie of door zelf een programma te ontwerpen) en ten slotte de gevonden oplossing in het oorspronkelijke vakgebied geïnterpreteerd wordt.

Het algemene onderzoeksdoel van deze dissertatie is het in kaart brengen van zowel de vakdidactische aspecten van Computational Science (modelleren en simuleren) binnen het examenvak Informatica in het voortgezet onderwijs in Nederland, als ook de vakdidactische kennis van docenten over het onderwijzen van dit onderwerp. Daartoe wordt de indeling van Magnusson et al. (1999) gehanteerd waarbij de vakdidactische aspecten bij een bepaald onderwerp gekarakteriseerd worden aan de hand van de volgende componenten:

M1 leerdoelen behorend bij dit onderwerp
M2 begrip van leerlingen over dit onderwerp
M3 instructiestrategieën voor dit onderwerp
M4 toetsingsmethoden voor dit onderwerp

De vakdidactische aspecten van Computational Science en de vakdidactische kennis van docenten voor het onderwijzen van Computational Science worden aan de hand van de volgende onderzoeksvragen onderzocht:

1. Welke computational thinking-activiteiten vormen het proces van probleem oplossen behorend bij Computational Science? Deze vraag is gericht op het definiëren van leerdoelen voor Computational Science (M1).
2. Hoe kan leerlingenbegrip van modelleeractiviteiten[35] worden gekarakteriseerd aan de hand van hun leerbehoeften en de moeilijkheden die ze ervaren? (M2)
3. Wat zijn de kenmerken van een valide en betrouwbaar toetsinstrument om Computational Science te toetsen? (M4).
4. Hoe kan pedagogical content knowledge (PCK) van leraren ten aanzien van Computational Science worden beschreven aan de hand van de kenniscomponenten M1 tot en met M4?

35 De begrippen Computational Science, modelleren en simuleren kunnen in deze tekst als synoniemen gezien worden.

8.2 Bevindingen

8.2.1 Operationele Definitie van Computational Science (vraag 1)

Hoofdstuk 3 beschrijft de eerste studie die ingaat op de eerste onderzoeksvraag: Welke computational thinking-activiteiten vormen het proces van probleem oplossen behorend bij Computational Science? Op basis van literatuurstudie is een eerste antwoord op deze vraag gekregen en die is verder verfijnd door leerlingen te observeren tijdens hun werk, door ze vragenlijsten te laten invullen en door individuele leerlingen te interviewen. Dit heeft geresulteerd in het volgende raamwerk met lijst van activiteiten die het proces van probleem oplossen behorend bij Computational Science vormen:

- Aangeven wat het *doel* is van het model
- Het *onderzoek* dat nodig is voor het bouwen van het model uitvoeren
- *Abstraheren* om onnodige details weg te laten
- Het probleem zodanig *formuleren* dat computers en ander gereedschap gebruikt kunnen worden om een computationele oplossing te vinden
- Opstellen van *vereisten en specificatie* voor die computationele oplossing
- Het *implementeren* van de computationele oplossing, oftewel: programmeren
- Het geïmplementeerde model *verifiëren* en *valideren*
- Het geïmplementeerde model gebruiken voor het *experimenteren, oftewel:* simulatie uitvoeren
- De uitkomsten verkregen door te experimenteren *analyseren*
- *Reflecteren* op het hele proces.

8.2.2 Leerlingen: begrip en moeilijkheden (vraag 2)

In de eerste en vierde studie (hoofdstukken 3 en 6) werden het begrip en de moeilijkheden die leerlingen ervaren tijdens hun werk aan modelleeropdrachten bestudeerd, om zo de tweede onderzoeksvraag te beantwoorden: Hoe kan leerlingenbegrip van modelleeractiviteiten worden gekarakteriseerd aan de hand van hun leerbehoeften en de moeilijkheden die ze ervaren? (M2) Er is vastgesteld dat veel leerlingen niet konden beslissen welk fenomeen te modelleren en moeite hadden met het vertalen van hun probleem naar computationele termen geschikt

om te programmeren. De meeste leerlingen is het gelukt om een programma te ontwerpen, maar vaak niet zoals zij dat wensten. Tijdens het testen wisten ze vaak niet of onverwacht gedrag van het programma door verkeerde aannames, programmeerfouten of door het gedrag behorend bij het gemodelleerde fenomeen werd veroorzaakt. Tijdens het programmeren van hun modellen werkten ze vaak volgens een incrementele trial-and-error-strategie. Weinig leerlingen voerden systematische en goed gedocumenteerde experimenten met hun modellen uit. Vaak werd het experimenteren en analyseren van verkregen uitkomsten vermengd met het construeren van modellen.

In de laatste studie (hoofdstuk 6) werden leerlingen geobserveerd terwijl ze leerden modelleren en werkten aan modelleeropdrachten aan de hand van lesmateriaal dat ik zelf heb geschreven. Focus lag voornamelijk op het begrip en de moeilijkheden rondom verificatie en validatie van hun modellen en deze werden beschreven in termen van:

- construeren van modellen
- testen
- reflecteren op geloofwaardigheid, overtuigingskracht, nauwkeurigheid van modellen en tevredenheid met modellen

Het bleek dat niet alle leerlingen het nodige onderzoek vooraf deden. Sommige maakten verkeerde aannames; er waren programmeerfouten; niet iedereen voerde experimenten systematisch uit; en sommige leerlingen waren tevreden met onrealistische modellen of leken de essentie van het modelleren niet helemaal begrepen te hebben.

8.2.3 Pedagogical Content Knowledge van Docenten (vraag 4)

In de tweede studie (hoofdstuk 4) werd de initiële PCK van Informaticadocenten rondom het modelleren en simuleren in kaart gebracht om een antwoord te geven op de vierde onderzoeksvraag: Hoe kan pedagogical content knowledge (PCK) van leraren ten aanzien van Computational Science worden beschreven aan de hand van de kenniscomponenten M1 tot en met M4? Wat betreft de leerdoelen van modelleren en simuleren, is het vastgesteld dat de beoogde leerdoelen voor leerlingen te verdelen zijn in conceptuele leerdoelen enerzijds, en motivatie gerelateerde en praktische leerdoelen anderzijds. In relatie tot het begrip van leerlingen zijn er drie aspecten te onderscheiden. Het betrof de benodigde voorkennis en vaardigheden om modellen te maken, de zaken die succesvol waren of aan het succes bijdragen zoals de ervaren relevatie van modellen, en ten slotte,

de zaken die leerlingen als moeilijk ervaren of voor problemen zorgen, zoals de variatie tussen de leerlingen in een klas of de moeite die leerlingen hebben met abstractie. Opvallend is dat sommige leraren niets wisten te vertellen over het begrip bij leerlingen. Wat betreft de instructiestrategieën zijn er vijf thema's: hoe leraren hun rol ervaren, opdrachten voor leerlingen, kenmerken van leerlingen om rekening mee te houden, organisatorische aspecten, en ten slotte, problemen en moeilijkheden. Er was een grote mate van overeenstemming rondom de gewenste instructiestrategieën: van docentgestuurd naar leerlinggestuurd, met als afsluiting een grote praktische opdracht die ook voor toetsing wordt gebruikt. Ten slotte, voor toetsing zijn er vier thema's: de toetsingsvorm, opdrachten die aan leerlingen worden gegeven, organisatorische aspecten en beoordelingscriteria. Er is overeenstemming over de gewenste vorm: een grote praktische opdracht. Echter, de beoordelingscriteria waren divers (zowel product als proces) en niet gespecificeerd in detail. Er is ook gekeken of er onderscheidende kenmerken van PCK van docenten kon identificeren met behulp van de vier componenten van PCK beschreven door Magnusson et al. (1999), maar dat bleek niet het geval.

8.2.4 Toetsinstrument voor Computational Science (vraag 3)

De bevindingen van de voorgaande studies hebben bijgedragen aan het beantwoorden van de derde onderzoeksvraag: Wat zijn de kenmerken van een valide en betrouwbaar toetsinstrument om Computational Science te toetsen? (M4). Met de inzichten die over de leerdoelen en geschikte onderwijsstrategieën uit voorgaande studies waren verkregen, is de eerste versie van lesmateriaal voor Computational Science ontwikkeld samen met een toetsinstrument in de vorm van een praktische opdracht met bijbehorende beoordelingsrubrics gebaseerd op SOLO[36] taxonomie. Deze taxonomie beschrijft de leeruitkomsten op vijf niveaus: prestructural[37], unistructural en multistructural — die als kwantitatief worden beschouwd — en relationeel en extended abstract — die duiden op een kwalitatieve verandering (Biggs & Tang, 2011). De praktische opdracht sluit nauw aan bij het raamwerk voor de ontwikkeling en het gebruik van computationele modellen dat ontwikkeld is als operationele definitie van Computational Science. Het bevat een reeks vragen en opdrachten die de leerlingen door het hele proces begeleiden. De leerlingen worden eerst gevraagd om het doel van het model te beschrijven (d.w.z. de onderzoeksvraag stellen) en eventueel onderzoek uit te voeren. Vervolgens

36 SOLO: Structure of the Observed Learning Outcome
37 Voor duidelijkheid werden hier de originele engelse termen gebruikt.

worden ze gevraagd om het model te ontwerpen, implementeren en valideren en daarmee te experimenten door het model uit te voeren, en om de uitkomsten te analyseren en de onderzoeksvraag te beantwoorden. Tenslotte worden ze gevraagd om te reflecteren op het hele proces. Dit resulteert in een portfolio met documentatie en het geïmplementeerd model. De rubrics classificeert de leeruitkomsten voor elk deel van het portfolio met behulp van SOLO-taxonomie. Het beoordelingsinstrument in de vorm van een praktische opdracht en bijbehorende rubrics op basis van de SOLO-taxonomie is betrouwbaar gebleken, zoals blijkt uit een hoog percentage interbeoordelaarsovereenkomsten. De validiteit ervan wordt bevestigd doordat de significante verschillen in prestatieniveaus van de HAVO leerlingen ten opzichte van de VWO leerlingen zichtbaar werden: zoals verwacht waren de prestatieniveaus van de VWO leerlingen significant hoger voor bijna alle criteria.

8.3 Wetenschappelijke Bijdrage

Dit onderzoeksproject levert verscheidene wetenschappelijke bijdragen.

De leerdoelen van het keuzethema Computational Science van het nieuwe examenprogramma voor Informatica zijn vanuit twee perspectieven in een raamwerk uitgewerkt: als een cyclisch proces gebaseerd op het modelleren binnen wiskunde en als een manier om CT en wetenschappelijk onderzoek te integreren vanuit het perspectief van Informaticaleerlingen. Deze methodologische toepassing is met name belangrijk omdat er groeiende interesse is voor de toepassing van CT in het moderne wetenschapsonderwijs.

Het begrip van leerlingen en moeilijkheden die ze ervaren tijdens het maken en gebruiken van computationele modellen zijn onderzocht vanuit het perspectief van Informatica waarbij een andere discipline de context biedt — dit in tegenstelling tot veel andere studies waar het leren van vakinhoud van de desbetreffende discipline centraal staat. Voor zover bekend, is dit de eerste keer dat het begrip van leerlingen en moeilijkheden die ze ervaren tijdens het maken en gebruik van computationele modellen voor hun wetenschappelijk onderzoek in deze context diepgaand zijn onderzocht, met name voor wat betreft het verifiëren en valideren van de gemaakte computationele modellen.

Het betrouwbaar en valide toetsinstrument is niet alleen handig voor gebruik in een onderwijssituatie, maar kan ook ingezet worden als *onderzoeksinstrument*

voor onderzoek naar de leerresultaten van leerlingen met betrekking tot computational thinking. In tegenstelling tot veel andere toetsinstrumenten met een nauwere focus, bekijkt dit instrument op een holistische wijze de gehele modelleercyclus.

Ten slotte is de PCK van Informaticadocenten betreffende Computational Science gedetailleerd in kaart gebracht. Hiermee is aangetoond dat het begrip PCK ook voor het onderzoek naar ideeën en kennis van Informaticadocenten ingezet kan worden.

De rode draad van de wetenschappelijke bijdrage van dit project kan gezien worden in het licht van het feit dat computational thinking — en zijn componenten modelleren en simuleren — een brug kan vormen tussen Informatica en een toepassingsdomein. Veel onderzoek in het Informaticaonderwijs is gewijd aan het programmeeronderwijs en beginnende programmeurs, en uit de literatuurstudie kwam geen onderzoek naar boven op het gebied van het gebruik van programmeervaardigheden om computationele modellen te ontwikkelen en die te gebruiken voor wetenschappelijk onderzoek binnen Informaticaonderwijs. De wetenschappelijke bijdrage van dit project aan de ontwikkeling van de theorie van het Informaticaonderwijs kan in dit licht gezien worden. Dit werk biedt een nieuw theoretisch kader voor empirisch onderzoek naar competenties van leerlingen en docenten op het gebied van het ontwerpen van computationele modellen, principes voor beoordeling en inzichten in het begrip van leerlingen met betrekking tot deze competenties. Bovendien is het de eerste studie waarin de PCK van de Informaticadocenten wordt onderzocht.

8.4 Praktische implicaties

De overkoepelende praktische bijdrage van dit onderzoeksproject kan het best geïllustreerd worden aan de hand van vergelijkbare verschuivingen van wetenschapsparadigma's (Hey et al., 2009). Veel grote theoretische wetenschappelijke resultaten — zoals klassieke mechanica of Lotka-Volterra-vergelijkingen die de dynamiek van biologische systemen beschrijven — werden mogelijk gemaakt door ontwikkelingen in de wiskunde (die op hun beurt vaak aangedreven werden door de behoeften van wetenschappers). De overeenkomstige verschuiving betekende dat de wetenschappelijke kennis in wiskundige termen kon worden beschreven, in plaats van beschrijvingen van

waarnemingen. Hiermee werd een wetenschappelijke paradigmaverschuiving teweeggebracht van de tekstuele beschrijving van waarnemingen naar het gebruik van wiskundige modellen. De ontwikkeling en het gebruik van dergelijke wiskundige representaties is vaak een complex proces. Bovendien is geavanceerde wiskundige kennis van differentiaal- en integraalberekening noodzakelijk; een actieve betrokkenheid bij wetenschappelijke activiteiten op deze manier ligt daarom vaak buiten het bereik van middelbare scholieren. De mogelijkheden die moderne computers tegenwoordig bieden maken het echter mogelijk om allerlei verschijnselen in computationele termen te beschrijven door simpelweg de kenmerken en het gedrag van de individuele elementen in het gemodelleerde systeem te beschrijven. Deze computationele beschrijvingen — d.w.z. computationele modellen — zijn uitvoerbaar en in de geest van weer een verdere wetenschappelijke paradigmaverschuiving. Hierbij verschuift de aandacht van de theoretische benadering naar de computationele benadering die zich op de simulatie van complexe verschijnselen richt door het gebruik van computationele modellen die uitvoerbaar zijn — in plaats van het maken van statische modellen. Dat betekent dat zelfs beginners zoals leerlingen hiermee aan de slag kunnen. Weintrop et al. (2016) merken op dat wetenschap steeds meer een computationele activiteit wordt. Zij beschouwen modelleren en simuleren als een van de belangrijkste categorieën van computational thinking voor wiskunde en wetenschap. Dit project levert het bewijs dat het mogelijk is om modelleren en simuleren binnen Informatica in het voortgezet onderwijs effectief te onderwijzen en zo leerlingen de gelegenheid te bieden om zelf met wetenschap bezig te zijn.

In hoofdstuk 2 is de geschiedenis en de situatie van Informatica in het voortgezet onderwijs in Nederland beschreven. De belangrijkste praktische bijdrage is daaraan gerelateerd: het draagt bij aan het onderwijs van Computational Science. De resultaten hebben bijgedragen aan het ontwerp van professionele ontwikkelingsactiviteiten voor leraren, waarbij zowel de aspecten van modellering en simulatie als de vakdidactiek daarvan aan bod komen. Bovendien heb ik als spin-off van dit project de bevindingen gebruikt om het ontwikkelen van lesmateriaal en een bijbehorende docentenhandleiding te begeleiden. Dit materiaal is nu in gangbare lesmethodes voor Informatica opgenomen. Daarnaast heb ik een uitgebreide cursus over Computational Science ontworpen die in het kader van bijscholing aan docenten Informatica wordt aangeboden. Het beoordelingsinstrument draagt bij aan de kwaliteit van de beoordeling op scholen. In een breder perspectief kunnen de bevindingen het onderwijs in andere

Informatica-onderwerpen beïnvloeden. Ze kunnen leraren informeren over begrip en moeilijkheden die ze van hun leerlingen kunnen verwachten. Daarnaast kan het de ontwikkeling van een vergelijkbaar holistisch beoordelingsinstrument bij andere kerndomeinen en keuzethema's van het Informaticaonderwijs inspireren.

Toen dit project begon, was het gericht op modelleren en simuleren binnen het Informaticaonderwijs. De hoop was dat het zou bijdragen aan het bouwen van bruggen naar andere vakken en dat het leerlingen en hun leraren zou stimuleren om samenwerking buiten de grenzen van hun disciplines te zoeken en zo ten goede zou komen aan het leren van alle betrokken disciplines. Nu kan er met tevredenheid vastgesteld worden dat computational thinking steeds meer aandacht krijgt, zoals bijvoorbeeld bij de voorstellen voor curriculumvernieuwing van curriculum.nu, en dat dit werk dus steeds relevanter wordt. Het krijgt steeds meer een centrale rol in het wetenschapsonderwijs (Park & Green, 2019), met name in verband met computationele modellen (Pears et al., 2019).

Als men buiten het Informaticaonderwijs kijk, dan is te zien dat het effectief gebruiken en begrijpen van computermodellen niet per se vereist dat leerlingen deze zelf ontwikkelen. Daardoor blijkt dit werk ook buiten het Informaticaonderwijs relevant te zijn. De bevindingen zijn ook van toepassing op het ontwerp van instructiestrategieën voor jongere leerlingen waarbij modellen gebruikt worden om het onderwijs van andere vakken te verrijken — zoals bijvoorbeeld in het TeaEdu4CT-project dat zich bezighoudt met opleiden van leraren in computational thinking bij STEAM[38]-vakken. Wanneer computational thinking in de context van een andere discipline toegepast wordt, kunnen leerlingen een bestaand computermodel (bijvoorbeeld over de verspreiding van een virus) gebruiken om simulaties uit te voeren, resultaten te analyseren en zowel wetenschappelijke als maatschappelijke consequenties te onderzoeken. Bovendien kunnen ze worden gestimuleerd om de aannames die ten grondslag liggen aan het model en de validiteit van het model te bespreken, en zo effectief deel te nemen aan de modelleercyclus en wetenschapspraktijken.

Dit voorbeeld is indicatief voor de curriculumvernieuwing die naar verwachting in Nederland plaats zal vinden, zoals besproken in hoofdstuk 2. Tegenwoordig wordt het belang van het leren van Informatica-gerelateerde vakinhoud erkend, zoals blijkt uit de voorstellen voor het nieuwe curriculum van het initiatief curriculum.nu. Dit initiatief pleit voor de introductie van een nieuw leerdomein Digitale Geletterdheid dat vier elementen bevat: ICT-vaardigheden,

38 STEAM: science, technology, engineering, art and mathematics

Mediawijsheid, Computational Thinking en Informatievaardigheden (Thijs et al., 2014). Het computational thinking element van digitale geletterdheid omvat een aantal Informaticaspecifieke onderwerpen, waaronder programmeren. Als de voorstellen van het curriculum.nu initiatief worden geaccepteerd en leiden tot de introductie van een nieuw curriculum voor het funderend onderwijs, zou dit het einde betekenen van een vrij unieke situatie waarin Nederland zich bevindt in vergelijking met veel moderne landen. In tegenstelling tot bijvoorbeeld Engeland (Barendsen et al., 2015), Denemarken (Caspersen & Nowack, 2013) of Litouwen (Dagienė & Stupuriene, 2016), kunnen Nederlandse leerlingen alleen Informatica volgen als ze naar gespecialiseerde ICT-beroepsopleidingen op het VMBO gaan, of Informatica als keuzevak volgen in de bovenbouw van HAVO en VWO — mits hun school dat vak aanbiedt en een leraar kan vinden. Gezien de huidige stand van zaken rond het Informaticaonderwijs in het Nederlandse primair en voortgezet onderwijs kan men alleen maar vaststellen dat het precair is, zoals aangegeven in hoofdstuk 2. Het is van groot belang om digitale geletterdheid in het hele Nederlandse basis- en voortgezet onderwijs in te voeren. Op die manier krijgen alle leerlingen eindelijk de kans om kennis te maken met Informatica.

8.5 Suggesties voor vervolgonderzoek

Dit onderzoeksproject vond grotendeels plaats vóór en gedeeltelijk tegelijk met de ontwikkeling van het nieuwe examenprogramma Informatica dat in 2019 werd ingevoerd. Hoewel de resultaten van dit project aan de ontwikkeling van het examenprogramma hebben bijgedragen, met name het keuzethema Computational Science, zijn de effecten en resultaten van de implementatie van het examenprogramma nog niet onderzocht. Het onderwijs kent momenteel vele initiatieven om computational thinking te onderwijzen, en daar is Computational Science een onderdeel van. Daarom wordt hier een aantal suggesties voor verder onderzoek gedaan.

Het onderzoek naar de initiële Pedagogical Content Knowledge (PCK) van de docenten met betrekking tot Computational Science werd uitgevoerd met een klein aantal docenten, wiens PCK op dat moment is in kaart gebracht werd. Voor vervolgonderzoek wordt voorgesteld om een grote diepgaande studie uit te voeren naar de ontwikkeling van de PCK van leraren met betrekking tot Computational Science terwijl ze hierover lesgeven en deelnemen aan relevante professionele

ontwikkelingsactiviteiten. Doel van een dergelijke studie is te onderzoeken hoe hun PCK evolueert naarmate ze ervaring opdoen met het lesgeven in Computational Science.

Er is lesmateriaal ontwikkeld om te helpen bij het onderzoek naar het begrip van begrip van leerlingen (M2) en beoordelingsmethoden (M4). De instructiestrategieën (M3) zelf zijn echter niet onderzocht. Vervolgonderzoek zou zich kunnen richten naar succesvolle instructiestrategieën voor Computational Science. Het als spin-off van dit project ontwikkeld lesmateriaal en het beoordelingsinstrument kunnen de basis vormen voor dit onderzoek.

Het beoordelingsinstrument op basis van de SOLO-taxonomie is in overeenstemming met de suggesties en behoeften van Informaticaleraren en voorziet in een holistische beoordeling van de leeruitkomsten van Computational Science. Vervolgonderzoek kan zich richten op de ontwikkeling van vergelijkbare beoordelingsinstrumenten, niet alleen gericht op de computationele concepten, maar ook op computationele praktijken en computationele perspectieven voor enerzijds andere leeruitkomsten van het schoolvak Informatica en anderzijds voor leeruitkomsten binnen andere disciplines waar computational thinking bij relevant is.

Dit project was gericht op onderzoek naar het onderwijzen van Computational Science binnen Informatica. Computational Science heeft tot doel Informaticaleerlingen gereedschappen, technieken en vaardigheden te bieden om modelleren en simuleren te gebruiken bij het verkennen van verschijnselen in verschillende wetenschappelijke disciplines buiten Informatica. Vervolgonderzoek kan drie lijnen volgen. Ten eerste, door de oorspronkelijke onderzoekslijn verder te vervolgen. Gezien het feit dat computationele modellen grote hoeveelheden data kunnen produceren, wordt er een onderzoekslijn voorgesteld in het kielzog van een andere wetenschappelijke paradigmaverschuiving — van een computationele benadering die complexe verschijnselen modelleert en simuleert naar een benadering die zich concentreert op het exploreren van data (Hey et al., 2009). Door de praktijken van data science die computational thinking en wiskundig denken samenbrengen, zouden de leerlingen die modellen ontwikkelen en simulaties uitvoeren de gelegenheid krijgen om de gegevens die door hun simulaties worden geproduceerd grondig te analyseren. Aangezien dit soort activiteiten slechts marginaal plaats vond binnen dit project, is verder onderzoek naar dit specifieke probleem op zijn plaats. Ten tweede, door breder dan alleen Informaticaonderwijs te kijken, ontstaat er een nieuw uitgangspunt—

het perspectief van computational thinking in context. Vervolgonderzoek kan zich richten op het begrip, de uitdagingen en de moeilijkheden van leerlingen met betrekking tot het leren van de vakinhoud waarvoor ze modellen maken en simulaties uitvoeren als beschreven in de leerdoelen van Computational Science. Ten derde, in het verlengde van de vorige suggestie, kunnen vakdidactische aspecten van het onderwijzen van digitale geletterdheid onderzocht worden, en in het bijzonder de component computational thinking — als het een onderdeel wordt van het curriculum voor funderend onderwijs in Nederland.

References

References

Adriaens, H., Fontein, P., Uijl, M. D., & Vos, K. D. (2016). De toekomstige arbeidsmarkt voor onderwijspersoneel po, vo en mbo 2015-2025, Update november 2016. CentERdata.

Aiken, J. M., Caballero, M. D., Douglas, S. S., Burk, J. B., Scanlon, E. M., Thoms, B. D., & Schatz, M. F. (2012). Understanding student computational thinking with computational modeling. arXiv preprint arXiv:1207.1764.

Akker, J. V. den, Gravemeijer, K., McKenney, S., & Nieveen, N. (2006). Educational design research. Taylor & Francis.

Allan, W., Coulter, B., Denner, J., Erickson, J., Lee, I., Malyn-Smith, J., & Martin, F. (2010). Computational Thinking for Youth White Paper. Walt Allan. http://itestlrc.edc.org/resources/computational-thinking-youth-white-paper

Alturki, R. A. (2016). Measuring and Improving Student Performance in an Introductory Programming Course. Informatics in Education-An International Journal, 15(2), 183–204.

Amouroux, E., Gaudou, B., Desvaux, S., & Drogoul, A. (2010). Odd: A promising but incomplete formalism for individual-based model specification. Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2010 IEEE RIVF International Conference on, 1–4.

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 computational thinking curriculum framework: Implications for teacher knowledge. Journal of Educational Technology & Society, 19(3).

Bakema, G., Zwart, J. P., & Lek, H. van der. (2002). Fully Communication Oriented Information Modeling, FCO-IM. Ten Hagen & Stam.

Barendsen, E., & Bruggink, M. (2019). Het volle potentieel van de computer leren benutten: Over informatica en computational thinking. In Van Twaalf tot Achttien.

Barendsen, E., & Tolboom, J. (2016). Advisory report (intended) curriculum for informatics for upper secondary education. SLO.

Barendsen, E., & Zwaneveld, B. (2010). Informatica in het Voortgezet Onderwijs Voorstel voor vakvernieuwing [Notitie voor KNAW].

Barendsen, E., Grgurina, N., & Tolboom, J. (2016). A New Informatics Curriculum for Secondary Education in The Netherlands. International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, 105–117.

Barendsen, E., Mannila, L., Demo, B., Grgurina, N., Izu, C., Mirolo, C., Sentance, S., Settle, A., & Stupurienė, G. (2015). Concepts in K-9 Computer Science Education. Proceedings of the 2015 ITiCSE on Working Group Reports, 85–116.

Basawapatna, A., Repenning, A., & Lewis, C. H. (2013). The simulation creation toolkit: An initial exploration into making programming accessible while preserving computational thinking. Proceeding of the 44th ACM technical symposium on Computer science education, 501–506.

Basu, S., Biswas, G., Sengupta, P., Dickes, A., Kinnebrew, J. S., & Clark, D. (2016). Identifying middle school students' challenges in computational thinking-based science learning. Research and Practice in Technology Enhanced Learning, 11(1), 13. https://doi.org/10.1186/s41039-016-0036-2

Basu, S., Dickes, A., Kinnebrew, J. S., Sengupta, P., & Biswas, G. (2013). CTSiM: A Computational Thinking Environment for Learning Science through Simulation and Modeling. The 5th International Conference on Computer Supported Education.

Basu, S., Dukeman, A., Kinnebrew, J., Biswas, G., & Sengupta, P. (2014). Investigating student generated computational models of science. Proceedings of the 11th International Conference of the Learning Sciences (ICLS 2014), 1097–1104.

Basu, S., McElhaney, K. W., Grover, S., Harris, C. J., & Biswas, G. (2018). A principled approach to designing assessments that integrate science and computational thinking. International Society of the Learning Sciences, Inc.[ISLS].

Bauer, B., Muller, J. P., & Odell, J. (2001). Agent UML: A formalism for specifying multiagent software systems. International journal of software engineering and knowledge engineering, 11(03), 207–230.

Baumert, J., Kunter, M., Blum, W., Brunner, M., Voss, T., Jordan, A., Klusmann, U., Krauss, S., Neubrand, M., & Tsai, Y. M. (2010). Teachers' mathematical knowledge, cognitive activation in the classroom, and student progress. American Educational Research Journal, 47(1), 133.

Bennett, J., & Holman, J. (2002). Context-Based Approaches to the Teaching of Chemistry: What are They and What Are Their Effects? (J. K. Gilbert, O. De Jong, R. Justi, D. F. Treagust, & J. H. Van Driel, Red.; pp. 165–184). Kluwer.

Bergervoet, P., Boon, P., Commandeur, K., Smeets, D., Van, der H., & Vorstenbosch, P. (2001). Informatica (2. Ed.). Edu'Actief.

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. Computers & Education, 72, 145–157.

Bielik, T., Fonio, E., Feinerman, O., Duncan, R. G., & Levy, S. T. (2021). Working Together: Integrating Computational Modeling Approaches to Investigate Complex Phenomena. Journal of Science Education and Technology, 30(1), 40–57.

Biesta, G. J. (2015). Good education in an age of measurement: Ethics, politics, democracy. Routledge.

Biggs, J., & Tang, C. (2011). Teaching for quality learning at university. McGraw-Hill International.

Blikstein, P., & Moghadam, S. H. (2019). Computing Education: Literature Review and Voices from the Field (S. Fincher & A. V. Robins, Red.; pp. 56–78). Cambridge University Press.

Blikstein, P., & Wilensky, U. (2009). An Atom is Known by the Company it Keeps: Content, Representation and Pedagogy within the Epistemic Revolution of the Complexity Sciences.

Borshchev, A. (2013). The big book of simulation modeling: Multimethod modeling with AnyLogic 6. AnyLogic North America.

Bort, H., & Brylow, D. (2013). CS4Impact: Measuring computational thinking concepts present in CS4HS participant lesson plans. Proceeding of the 44th ACM technical symposium on Computer science education, 427–432. https://doi.org/10.1145/2445196.2445323

Boulay, B. D. (1986). Some difficulties of learning to program. Journal of Educational Computing Research, 2(1), 57–73.

Brade, D. (2004). A generalized process for the verification and validation of models and simulation results. A generalized process for the verification and validation of models and simulation results.

Brennan, K., & Resnick, M. (2012). New Frameworks for Studying and Assessing the Development of Computational Thinking. Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada.

Brodjanac, P., Bubica, N., Kralj, L., Markucic, Z., Mirkovic, M., Rubie, M., Sudarevic, D., Czwyk, A., Mari, C., Hrzica, V., & Vuk, B. (2016). Nacionalni kurikulum nastavnoga predmeta informatika. http://www.kurikulum.hr/wp-content/uploads/2016/03/Informatika.pdf; http://www.kurikulum.hr/wp-content/uploads/2016/03/Informatika.pdf

Bungartz, H.-J., Zimmer, S., Buchholz, M., Pflüger, D., Le Borne, S., & Le Borne, R. (2014). Modeling and simulation: An application-oriented introduction/by Hans-Joachim Bungartz, Stefan Zimmer, Martin Buchholz, Dirk Pflüger; translated by Sabine Le Borne, Richard Le Borne.

Caballero, M. D., Kohlmyer, M. A., & Schatz, M. F. (2012). Implementing and assessing computational modeling in introductory mechanics. Physical Review Special Topics - Physics Education Research, 8(2), 020106. https://doi.org/10.1103/PhysRevSTPER.8.020106

Çakıroğlu, Ü., & Kiliç, S. (2020). Assessing teachers' PCK to teach computational thinking via robotic programming. Interactive Learning Environments, 0(0), 1–18. https://doi.org/10.1080/10494820.2020.1811734

Carley, K. M. (1996). Validating computational models. Paper available at http://www.casos.cs.cmu.edu/publications/papers.php.

Carlson, J., & Daehler, K. R. (2019). The refined consensus model of pedagogical content knowledge in science education (pp. 77–92). Springer.

Carnegie Mellon Center for Computational Thinking. (2010). https://www.cs.cmu.edu/~CompThink/

Carson, I. I., & John, S. (2004). Introduction to modeling and simulation. Proceedings of the 36th conference on Winter simulation, 9–16.

Caspersen, M. E., & Nowack, P. (2013a). Computational Thinking and Practice—A Generic Approach to Computing in Danish High Schools.

Caspersen, M. E., & Nowack, P. (2013b). Model–Based Thinking & Practice. Centre for Science Education, Aarhus University.

Castro, F. E. V., & Fisler, K. (2017). Designing a multi-faceted SOLO taxonomy to track program design skills through an entire course. Proceedings of the 17th Koli Calling Conference on Computing Education Research, 10–19.

Cateté, V., Lytle, N., Dong, Y., Boulden, D., Akram, B., Houchins, J., Barnes, T., Wiebe, E., Lester, J., & Mott, B. (2018). Infusing computational thinking into middle grade science classrooms: Lessons learned. Proceedings of the 13th Workshop in Primary and Secondary Computing Education, 1–6.

Center for Scientific Workshops in All Disciplines—Computing in Secondary Education. (2014). https://www.lorentzcenter.nl/computing-in-secondary-education.html

Chalmers, C. (2018). Robotics and computational thinking in primary school. International Journal of Child-Computer Interaction, 17, 93–100. https://doi.org/10.1016/j.ijcci.2018.06.005

Cohen, L., Manion, L., & Morrison, K. R. B. (2007). Research methods in education, Sixth edition. Routledge.

College voor Toetsen en Examens. (1998). Vaststelling examenprogramma's vwo/havo (Gele katern). https://www.examenblad.nl/publicatie/19980506/vaststelling-examenprogramma-s-vwo/2009?regime=hfregts&horizon=vg41h1h6n8tf

Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., Young, P. R., & Denning, P. J. (1989). Computing as a Discipline. Communications of the ACM, 32(1), 9–23.

Computational thinking. (2020). [Overzichtspagina]. SLO. https://slo.nl/vakportalen/vakportaal-digitale-geletterdheid/computational-thinking/

CS For All. (2016). https://www.csforall.org/; https://www.csforall.org/; https://www.csforall.org/

CSTA Computational Thinking Task Force. (2011). Operational Definition of Computational Thinking for K–12 Education (Vol. 2013).

Curzon, P., Dorling, M., Ng, T., Selby, C., & Woollard, J. (2014). Developing computational thinking in the classroom: A framework.

Czerkawski, B. (2013). Instructional design for computational thinking. Proceedings of Society for Information Technology & Teacher Education International Conference, 10–17.

Czerkawski, B., & Xu, L. (2012). Computational Thinking and Educational Technology. World Conference on Educational Multimedia, Hypermedia and Telecommunications, 2012, 2607–2610.

Dagienė, V., & Sentance, S. (2016). It's computational thinking! Bebras tasks in the curriculum. International conference on informatics in schools: Situation, evolution, and perspectives, 28–39.

Dagienė, V., & Stupuriene, G. (2016). Informatics concepts and computational thinking in K-12 education: A Lithuanian perspective. Journal of Information Processing, 24(4), 732–739.

Dagienė, V., & Stupuriene, G. (2016a). Bebras-a sustainable community building model for the concept based learning of informatics and computational thinking. Informatics in Education, 15(1), 25.

Dagienė, V., & Stupuriene, G. (2016b). Informatics concepts and computational thinking in K-12 education: A Lithuanian perspective. Journal of Information Processing, 24(4), 732–739.

Davies, S. (2008). The effects of emphasizing computational thinking in an introductory programming course. 2008 38th Annual Frontiers in Education Conference, T2C-3-T2C-8.

De Araujo, A. L. S. O., Andrade, W. L., & Guerrero, D. D. S. (2016). A systematic mapping study on assessing computational thinking abilities. 2016 IEEE frontiers in education conference (FIE), 1–9.

Denning, P. J. (2009). The profession of IT Beyond computational thinking. Communications of the ACM, 52(6), 28–30.

Dirks, F., & Tolboom, J. (2000). CODI curriculum from the perspective of the teacher's practice. Tinfon. Tinfon, 9, 104–107.

Downes, T. (2007). Informatics Education: A Case study of the confusions and complexities of the intended and enacted curriculum in NSW Secondary Schools.

DUO. (2018). Leerlingen in het voortgezet onderwijs (Vol. 2018). https://duo.nl/open_onderwijsdata/databestanden/vo/leerlingen/

Dwyer, H., Boe, B., Hill, C., Franklin, D., & Harlow, D. (2013). Computational Thinking for Physics: Programming Models of Physics Phenomenon in Elementary School.

Edo, S. I., Putri, R. I. I., & Hartono, Y. (2013). Investigating secondary school students' difficulties in modeling problems PISA-Model Level 5 and 6. Journal on mathematics Education, 4(1), 41–58.

Eraslan, A., & Kant, S. (2015). Modeling Processes of 4th-Year Middle-School Students and the Difficulties Encountered. Educational Sciences: Theory & Practice, 15(3), Article 3. https://doi.org/10.12738/estp.2015.3.2556

Every Student Succeeds Act (ESSA), (2015). https://www.ed.gov/essa?src=rn; https://www.ed.gov/essa?src=rn; https://www.ed.gov/essa?src=rn

Fletcher, G. H. L., & Lu, J. J. (2009). Education Human computing skills: Rethinking the K-12 experience. Communications of the ACM, 52(2), 23–25.

Furber, S. (2012). Shut down or restart? The way forward for computing in UK schools. The Royal Society, London.

Gal-Ezer, J. (1995). Computer science teachers' certification program. Computers & Education, 25(3), 163–168.

Gal-Ezer, J., Beeri, C., Harel, D., & Yehudai, A. (1995). A high school program in computer science. Computer, 28(10), 73–80.

Gander, W., Petit, A., Berry, G., Demo, B., Vahrenhold, J., McGettrick, A., Boyle, R., Mendelson, A., Stephenson, C., Ghezzi, C., & others. (2013). Informatics education: Europe cannot afford to miss the boat. Report of the joint Informatics Europe & ACM Europe Working Group on Informatics Education.

Gendreau Chakarov, A., Recker, M., Jacobs, J., Van Horne, K., & Sumner, T. (2019). Designing a Middle School Science Curriculum that Integrates Computational Thinking and Sensor Technology. Proceedings of the 50th ACM Technical Symposium on Computer Science Education, 818–824.

Gilbert, J. K. (2006). On the nature of "context" in chemical education. International Journal of Science Education, 28(9), 957–976.

Gilbert, J. K., & Justi, R. (2016). Modelling-based teaching in science education (Vol. 9). Springer.

Ginjaar-Maas, N. J. (1994). De Tweede Fase vernieuwt: Scharnier tussen basisvorming en hoger onderwijs (Rep. No. 2). Stuurgroep Profiel Tweede Fase Voortgezet Onderwijs.

Glass, R. L. (2006). Call it problem solving, not computational thinking. Communications of the ACM, 49(9), 13–13.

Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013a). Computational thinking in educational activities: An evaluation of the educational game light-bot. Proceedings of the 18th ACM conference on Innovation and technology in computer science education, 10–15. https://doi.org/10.1145/2462476.2466518

Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013b). First Year Student Performance in a Test for Computational Thinking. Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference, 271–277. https://doi.org/10.1145/2513456.2513484

Granger, Chris. (2015). Chris Granger—Coding is not the new literacy. https://www.chris-granger.com/2015/01/26/coding-is-not-the-new-literacy/

Grgurina, N. (2013). Computational Thinking in Dutch Secondary Education. Informatics in Schools: Local Proceedings of the 6th International Conference ISSEP 2013–Selected Papers, 119.

Grgurina, N., & Tolboom, J. (2008). The First Decade of Informatics in Dutch High Schools. Informatics in Education, 7(1), 55–74.

Grgurina, N., Barendsen, E., Suhre, C., Veen, K. van, & Zwaneveld, B. (2017). Investigating Informatics Teachers' Initial Pedagogical Content Knowledge on Modeling and Simulation. International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, 65–76.

Grgurina, N., Barendsen, E., Suhre, C., Veen, K. van, & Zwaneveld, B. (2018). Assessment of Modeling Projects in Informatics Class (V. Dagienė & E. Jasute, Red.; pp. 571–577). Vilnius University.

Grgurina, N., Barendsen, E., Suhre, C., Zwaneveld, B., & Veen, K. van. (2018). Assessment of modeling and simulation in secondary computing science education. 7.

Grgurina, N., Barendsen, E., Veen, K. van, Suhre, C., & Zwaneveld, B. (2015). Exploring Students' Computational Thinking Skills in Modeling and Simulation Projects: A Pilot Study. Proceedings of the Workshop in Primary and Secondary Computing Education, 65–68.

Grgurina, N., Barendsen, E., Zwaneveld, B., Veen, K. van, & Stoker, I. (2014a). Computational Thinking Skills in Dutch Secondary Education: Exploring Pedagogical Content Knowledge. Proceedings of the 14th Koli Calling International Conference on Computing Education Research, 173–174.

Grgurina, N., Barendsen, E., Zwaneveld, B., Veen, K. van, & Stoker, I. (2014b). Computational Thinking Skills in Dutch Secondary Education: Exploring Teacher's Perspective. Proceedings of the 9th Workshop in Primary and Secondary Computing Education, 124–125.

Grgurina, N., Barendsen, E., Zwaneveld, B., Veen, K. van, & Suhre, C. (2016). Defining and Observing Modeling and Simulation in Informatics. International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, 130–141.

Grgurina, N., Tolboom, J., & Barendsen, E. (2018). The Second Decade of Informatics in Dutch Secondary Education. 271–282.

Grgurina, N., van der Veen, R., & Velthuizen, V. (2019). R. Computational science · Informatica 2019. Domein R: Computational Science. https://ieni.github.io/inf2019/themas/r-computational-science

Griffin, J., Pirmann, T., & Gray, B. (2016). Two teachers, two perspectives on CS principles. Proceedings of the 47th ACM technical symposium on computing science education, 461–466.

Grimm, V., Berger, U., Bastiansen, F., Eliassen, S., Ginot, V., Giske, J., Goss-Custard, J., Grand, T., Heinz, S. K., & Huse, G. (2006). A standard protocol for describing individual-based and agent-based models. Ecological Modelling, 198(1–2), 115–126.

Grimm, V., Berger, U., DeAngelis, D. L., Polhill, J. G., Giske, J., & Railsback, S. F. (2010). The ODD protocol: A review and first update. Ecological Modelling, 221(23), 2760–2768.

Grossman, P., Schoenfeld, A., & Lee, C. (2005). Teaching subject matter. Preparing teachers for a changing world: What teachers should learn and be able to do, 201–231.

Grover, S. (2011). Robotics and Engineering for Middle and High School Students to Develop Computational Thinking. annual meeting of the American Educational Research Association, New Orleans, LA.

Grover, S., & Pea, R. (2018). Computational Thinking: A competency whose time has come. Computer Science Education: Perspectives on Teaching and Learning in School, 19–38.

Guerra, V., Kuhnt, B., & Blöchliger, I. (2012). Informatics at school-Worldwide. An international exploratory study about informatics as a subject at different school levels.

Guzdial, M. (2015). Growing CS Ed through Schools of Ed, and CT is Unlikely: Report from Oldenburg. In Computing Education Research Blog. https://computinged.wordpress.com/2015/07/13/growing-cs-ed-through-schools-of-ed-report-from-oldenburg/; https://computinged.wordpress.com/2015/07/13/growing-cs-ed-through-schools-of-ed-report-from-oldenburg/

Guzdial, M. (2018). Maybe there's more than one kind of Computational Thinking, but that makes research difficult. In Computing Education Research Blog. https://computinged.wordpress.com/2018/12/07/maybe-theres-more-that-one-kind-of-computational-thinking/; https://computinged.wordpress.com/2018/12/07/maybe-theres-more-that-one-kind-of-computational-thinking/

Guzdial, M. (2019). Computing for Other Disciplines (S. A. Fincher & A. V. Robins, Red.; pp. 584–605). Cambridge University Press.

Guzdial, M., & Boulay, B. du. (2019). The History of Computing Education Research (S. A. Fincher & A. V. Robins, Red.; pp. 11–39). Cambridge University Press.

Hacquebard, A. E. N., Hartsuijker, A., Brinkkemper, J. N., van, der H., Hogenbirk, P. G., Ikkersheim, D. C., Smeets, D. A. J., & Vorstenbosch, P. A. M. (1995). Advies Examenprogramma's havo/vwo: Informatica. Stuurgroep Profiel Tweede Fase. https://research.utwente.nl/en/publications/advies-examenprogrammas-havovwo-informatica(21ed9ef1-da52-47fb-93ac-9aea8881555b).html; https://research.utwente.nl/en/publications/advies-examenprogrammas-havovwo-informatica(21ed9ef1-da52-47fb-93ac-9aea8881555b).html

Hacquebard, A. E. N., Zwaneveld, B., Dijk, B. van, Leeuwen, H. van, & Timmers, J. (2005). Keuzevak Informatica in de Tweede Fase HAVO en VWO, Opstap naar de kennismaatschappij. CODI.Ref Type, Pamphlet.

Hallström, J., & Schönborn, K. J. (2019). Models and modelling for authentic STEM education: Reinforcing the argument. International Journal of STEM Education, 6(1), 22.

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A multidisciplinary approach towards computational thinking for science majors. Proceedings of the 40th ACM technical symposium on Computer science education, 183–187.

Hartsuijker, A., Westland, F. (2004). Vakdossiers 2003 Informatica. SLO.

Hartsuijker, A., van Dijk, B., Timmers, J., Zwaneveld, B. (2003). Vakdossiers 2002 Informatica. SLO.

Hartsuijker, A., Kuipers, T., Andries, de R., Dirk-Jan, van de P., Grgurina, N., Nowak, K., & Woudt, R. (2001). Vakdossiers 2001 Informatica. SLO.

Hartsuijker, A., M.A.G, E. van D., Kuipers, T. (2001). Vakdossiers 2000: Informatica.

Hemmendinger, D. (2010). A plea for modesty. ACM Inroads, 1(2), 4–7.

Henderson, P. B. (2009). Ubiquitous computational thinking. Computer, 42(10), 100–102.

Henze, I., & Barendsen, E. (2019). Unravelling student teachers' PCK development and the influence of personal factors using authentic data sources. In A. Hume, R. Cooper, & A. Borowski (Eds.), Repositioning Pedagogical Content Knowledge in teachers' knowledge for teaching science (pp. 201–221). Springer.

Henze, I., Driel, J. H. van, & Verloop, N. (2007). Science teachers' knowledge about teaching models and modelling in the context of a new syllabus on public understanding of science. Research in Science Education, 37(2), 99–122.

Henze, I., Driel, J. H. van, & Verloop, N. (2008). Development of experienced science teachers' pedagogical content knowledge of models of the solar system and the universe. International Journal of Science Education, 30(10), 1321–1342.

Heuvelink, A., Leijtens, R., & Loots, M. (2008). Module AI. Het is Amsterdam.

Hey, A. J., Tansley, S., & Tolle, K. M. (2009). The fourth paradigm: Data-intensive scientific discovery (Vol. 1). Microsoft research Redmond, WA.

Howland, K., Good, J., & Nicholson, K. (2009). Language-based support for computational thinking. 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 147–150.

Hu, C. (2011). Computational thinking: What it might mean and what we might do about it. Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, 223–227. https://doi.org/10.1145/1999747.1999811

Hubwieser, P. (2012). Computer Science Education in Secondary Schools–The Introduction of a New Compulsory Subject. ACM Transactions on Computing Education (TOCE), 12(4), 16.

Hubwieser, P., Magenheim, J., Muhling, A., & Ruf, A. (2013). Towards a conceptualization of pedagogical content knowledge for computer science. Proceedings of the ninth annual international ACM conference on International computing education research, 1–8. https://doi.org/10.1145/2493394.2493395

Hulsen, M., Wartenbergh-Cras, F., Smets, E., Uerz, D., van, der N., & Sontag, L. (2005). ICT in Cijfers -ICT-onderwijsmonitor studiejaar 2004/2005. IVA-ITS.

Jansen, M. (2007). The Education System in the Netherlands 2007. Dutch Eurydice Unit Ministry of Education, Culture and Science.

Jones, E. (2011). The Trouble with Computational Thinking. https://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/JonesCTOnePager.pdf

Justi, R., & Gilbert, J. K. (2002). Science teachers' knowledge about and attitudes towards the use of models and modelling in learning science. International Journal of Science Education, 24(12), 1273–1292.

Justi, R., & Gilbert, J. K. (2003). Teachers' views on the nature of models. International Journal of science education, 25(11), 1369–1386.

Kafai, Y. B., & Burke, Q. (2013). The social turn in K-12 programming: Moving from computational thinking to computational participation. Proceeding of the 44th ACM technical symposium on computer science education, 603–608.

Kafai, Y. B., Searle, K., Kaplan, E., Fields, D., Lee, E., & Lui, D. (2013). Cupcake cushions, scooby doo shirts, and soft boomboxes: E-textiles in high school to promote computational concepts, practices, and perceptions. Proceeding of the 44th ACM technical symposium on Computer science education, 311–316.

Kafura, D., & Tatar, D. (2011). Initial experience with a computational thinking course for computer science students. Proceedings of the 42nd ACM technical symposium on Computer science education, 251–256.

Kirschner, P. A., & Merriënboer, J. V. (2008). Ten steps to complex learning a new approach to instruction and instructional design.

KNAW. (2012). Digitale geletterdheid in het voortgezet onderwijs. Koninklijke Nederlandse Akademie van Wetenschappen.

Koh, K. H., Nickerson, H., Basawapatna, A., & Repenning, A. (2014). Early validation of computational thinking pattern analysis. Proceedings of the 2014 conference on Innovation & technology in computer science education, 213–218.

Kolikant, Y. B.-D. (2005). Students' alternative standards for correctness. Proceedings of the first international workshop on Computing education research, 37–43.

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. Theory into practice, 41(4), 212–218.

Law, A. M. (2009). How to Build Valid and Credible Simulation Models. In M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, & R. G. Ingalls (Red.), Proceedings of the 2009 Winter Simulation Conference.

Law, A. M. (2015). Simulation Modeling and Analysis Fifth Edition. McGraw-Hill.

Lee, E., Brown, M. N., Luft, J. A., & Roehrig, G. H. (2007). Assessing beginning secondary science teachers' PCK: Pilot year results. School Science and Mathematics, 107(2), 52–60.

Lee, I., Grover, S., Martin, F., Pillai, S., & Malyn-Smith, J. (2020). Computational thinking from a disciplinary perspective: Integrating computational thinking in K-12 science, technology, engineering, and mathematics education. Journal of Science Education and Technology, 29(1), 1–8.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L. (2011). Computational thinking for youth in practice. ACM Inroads, 2(1), 32–37.

Liberman, N., Kolikant, Y. B.-D., & Beeri, C. (2012). 'Regressed experts' as a new state in teachers' professional development: Lessons from Computer Science teachers' adjustments to substantial changes in the curriculum. Computer Science Education, 22(3), 257–283.

Lockwood, J., & Mooney, A. (2017). Computational Thinking in Education: Where does it fit? A systematic literary review. arXiv preprint arXiv:1703.07659.

Louca, L. T., Zacharia, Z. C., Michael, M., & Constantinou, C. P. (2011). Objects, entities, behaviors, and interactions: A typology of student-constructed computer-based models of physical phenomena. Journal of Educational Computing Research, 44(2), 173–201.

Loughran, J., Mulhall, P., & Berry, A. (2004). In search of pedagogical content knowledge in science: Developing ways of articulating and documenting professional practice. Journal of Research in Science Teaching, 41(4), 370–391.

Lu, J. J., & Fletcher, G. H. L. (2009). Thinking about computational thinking. Proceedings of the 40th ACM technical symposium on Computer science education, 260–264.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? Computers in Human Behavior, 41, 51–61. http://dx.doi.org.proxy-ub.rug.nl/10.1016/j.chb.2014.09.012

Maaß, K. (2006). What are modelling competencies? ZDM, 38(2), 113–142.

Magnusson, S., Krajcik, J., & Borko, H. (1999). Nature, sources, and development of pedagogical content knowledge for science teaching (J. Gess-Newsome & N. G. Lederman, Red.; pp. 95–132). Kluwer.

Malmi, L., Sheard, J., Bednarik, R., Helminen, J., Korhonen, A., Myller, N., Sorva, J., & Taherkhani, A. (2010). Characterizing research in computing education: A preliminary analysis of the literature. 3–12.

Malyn-Smith, J., Lee, I. A., Martin, F., Grover, S., Evans, M. A., & Pillai, S. (2018). Developing a framework for computational thinking from a disciplinary perspective. Conference Proceedings of the International Conference on Computational Thinking Education, 182–186.

Martin, F. G. (2012). Personal communication. http://www.cs.uml.edu/~fredm/

Martins-Pacheco, L. H., von Wangenheim, C. A. G., & da Cruz Alves, N. (2019). Assessment of Computational Thinking in K-12 Context: Educational Practices, Limits and Possibilities-A Systematic Mapping Study. Proceedings of the 11th International Conference on Computer Supported Education (CSEDU 2019), 1, 292–303.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning Computer Science Concepts with Scratch. Computer Science Education, 23(3), 239–264.

Meijer, H., Hart, T., & Reinders, H. (2001). Turing. ThiemeMeulenhoff.

MinOC&W: Omscholing informatica Cfi. In UITLEG Gele Katern nr. 7, vol. 14, pp. 22. (1998)

Müller, B., Balbi, S., Buchmann, C. M., Sousa, L. D., Dressler, G., Groeneveld, J., Klassert, C. J., Le, Q. B., Millington, J. D., & Nolzen, H. (2014). Standardised and transparent model descriptions for agent-based models: Current status and prospects. Environmental Modelling & Software, 55, 156–163.

Naylor, T. H., & Finger, J. M. (1967). Verification of computer simulation models. Management science, 14(2), 101.

Nicolaou, C. T., & Constantinou, C. P. (2014). Assessment of the modeling competence: A systematic review and synthesis of empirical research. Educational Research Review, 13, 52–73.

Odell, J. J., Parunak, H. V. D., & Bauer, B. (2000). Representing agent interaction protocols in UML. International Workshop on Agent-Oriented Software Engineering, 121–140.

Opdracht vernieuwingscommissie informatica 2014-2015. (2014). Ministry of Education, Netherlands.

Overveld, K. V., Borghuis, T., & Berkum, E. van. (2015). From Problems to Numbers and Back. Eindhoven University of Technology.

Papaevripidou, M., Nicolaou, C. T., & Constantinou, C. P. (2014). On defining and assessing learners' modeling competence in science teaching and learning. Annual Meeting of American Educational Research Association (AERA), Philadelphia, Pennsylvania, USA.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc.

Park, Y.-S., & Green, J. (2019). Bringing Computational Thinking into Science Education. Journal of the Korean Earth Science Society, 40(4), 340–352. https://doi.org/10.5467/JKESS.2019.40.4.340

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. New ideas in psychology, 2(2), 137–168.

Pears, A., Barendsen, E., Dagienė, V., Dolgopolovas, V., & Jasutė, E. (2019). Holistic STEAM education through computational thinking: A perspective on training future teachers. International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, 41–52.

Perković, L., Settle, A., Hwang, S., & Jones, J. (2010). A framework for computational thinking across the curriculum. Proceedings of the fifteenth annual conference on Innovation and technology in computer science education, 123–127.

Perrenet, J., & Zwaneveld, B. (2012). The many faces of the mathematical modeling cycle. Journal of Mathematical Modelling and Application, 1(6), 3–21.

Pfefferova, M. S. (2015). Computer Simulations and their Influence on Students' Understanding of Oscillatory Motion. Informatics in Education, 14(2), 279–289.

Polya, G. (2008). How to solve it: A new aspect of mathematical method. Princeton University Press.

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

Qin, H. (2009). Teaching computational thinking through bioinformatics to biology students. Proceedings of the 40th ACM technical symposium on Computer science education, 188–191.

Rahimi, E., Barendsen, E., & Henze, I. (2016). Typifying Informatics Teachers' PCK of Designing Digital Artefacts in Dutch Upper Secondary Education. International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, 65–77.

Rand, W., & Wilensky, U. (2006). Verification and validation through replication: A case study using Axelrod and Hammond's ethnocentrism model. North American Association for Computational Social and Organization Sciences (NAACSOS), 1–6.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. Computer science education, 13(2), 137–172.

Román-González, M., Moreno-León, J., & Robles, G. (2017). Complementary tools for computational thinking assessment. Proceedings of International Conference on Computational Thinking Education (CTE 2017), S. C Kong, J Sheldon, and K. Y Li (Eds.). The Education University of Hong Kong, 154–159.

Rumbaugh, J., Jacobson, I., & Booch, G. (2004). Unified modeling language reference manual, the. Pearson Higher Education.

Ruthmann, A., Heines, J. M., Greher, G. R., Laidler, P., & Saulters, I. I. (2010). Teaching computational thinking through musical live coding in scratch. Proceedings of the 41st ACM technical symposium on Computer science education, 351–355.

Saeli, M. (2012). Teaching Programming for Secondary School: A Pedagogical Content Knowledge Base Approach [PhD Thesis].

Saeli, Mara, Perrenet, J., Jochems, W. M. G., & Zwaneveld, B. (2012). Programming: Teachers and Pedagogical Content Knowledge in the Netherlands. Informatics in Education, 11(1), 81–114.

Sanders, L. R., Borko, H., & Lockard, J. D. (1993). Secondary science teachers' knowledge base when teaching science courses in and out of their area of certification. Journal of Research in Science Teaching, 30(7), 723–736.

Sargent, R. G. (2013). Verification and validation of simulation models. Journal of simulation, 7(1), 12–24.

Schmid, A. (2005). What is the Truth of Simulation? Journal of Artificial Societies and Social Simulation, 8(4).

Schmidt, V. (2006). Handreiking schoolexamen informatica havo/vwo. SLO, Enschede.

Schmidt, V. (2007). Vakdossier 2007 informatica (dossier of the subject computer science 2007)(Tech. Rep.). Enschede, The Netherlands: Slo, Stichting Leerplanontwikkeling.

School Education in France—Éduscol. Geraadpleegd 15 juni 2020, van https://eduscol.education.fr/pid26689/school-education-in-france.html

Selby, C. (2013, januari). Computational thinking: The developing definition. The 18th Annual Conference on Innovation and Technology in Computer Science Education. http://eprints.soton.ac.uk/346937/

Selby, C. C. (2014). How can the teaching of programming be used to enhance computational thinking skills? [PhD Thesis]. University of Southampton.

Sengupta, P., Dickes, A., & Farris, A. (2018). Toward a phenomenology of computational thinking in STEM education (pp. 49–72). Springer.

Sengupta, P., Kinnebrew, J. S., Basu, S., Biswas, G., & Clark, D. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. Education and Information Technologies, 1–30.

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

Sins, P. H. M., Savelsbergh, E. R., & Joolingen, W. R. van. (2005). The Difficult Process of Scientific Modelling: An analysis of novices' reasoning during computer-based modelling. International Journal of Science Education, 27(14), 1695–1721. https://doi.org/10.1080/09500690500206408

Society, P. board N. & P. board. (2005). Bridges between Nature and Society. http://www.minocw.nl/documenten/49152.pdf

Sturrock, D. T. (2015). Tutorial: Tips for Successful Practice of Simulation. In L. Yilmaz, W. K. V. Chan, I. Moon, M. K. Roeder, C. Macal, & D. Rosserri (Red.), Proceedings of the 2015 Winter Simulation Conference.

Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. Computers & Education, 148, 103798.

Taub, R., Armoni, M., & Ben-Ari, M. (2013). The Contribution of Computer Science to Learning Computational Physics (pp. 127–137). Springer.

Taub, R., Armoni, M., & Ben-Ari, M. (2014). Abstraction as a Bridging Concept Between Computer Science and Physics. Proceedings of the 9th Workshop in Primary and Secondary Computing Education, 16–19.

Teaching computer science in France: Tomorrow can't wait. (2013). Académie des Sciences.

Tedre, M., & Denning, P. J. (2016). The long quest for computational thinking. Proceedings of the 16th Koli Calling Conference on Computing Education Research, 24–27.

Tedre, M., Simon, & Malmi, L. (2018). Changing aims of computing education: A historical survey. Computer Science Education, 28(2), 158–186. https://doi.org/10.1080/08993408.2018.1486624

Tedre, M., Simon, & Malmi, L. (2018). Changing aims of computing education: A historical survey. Computer Science Education, 28(2), 158–186. https://doi.org/10.1080/08993408.2018.1486624

Thijs, A. M., Fisser, P., & Hoeven, M. V. D. (2014a). 21e eeuwse vaardigheden in het curriculum van het funderend onderwijs. SLO.

Thijs, A. M., Fisser, P., & Hoeven, M. V. D. (2014b). Digitale geletterdheid en 21e eeuwse vaardigheden in het funderend onderwijs: Een conceptueel kader (draft). SLO.

Thinking, N. R. C. (US) C. for the W. on C., & Council, N. R. (2010). Report of a workshop on the scope and nature of computational thinking. Natl Academy Pr.

Tolboom, J. (1999). The first year of the CODI curriculum. Informatie bulletin I&I, 9–11.

Tolboom, J., Kruger, J., & Grgurina, N. (2014). Informatica in de bovenbouw havo/vwo: Naar aantrekkelijk en actueel onderwijs in informatica. SLO.

Touretzky, D. S., Marghitu, D., Ludi, S., Bernstein, D., & Ni, L. (2013). Accelerating K-12 computational thinking using scaffolding, staging, and abstraction. Proceeding of the 44th ACM technical symposium on Computer science education, 609–614.

Tweede Fase Adviespunt. (2006). Veranderingen Tweede Fase en docenten: Volgen of meebeslissen (Vol. 2008).

Vaandrager, F. (2011). A First Introduction to uppaal. Deliverable no.: D5.12 Title of Deliverable: Industrial Handbook, 18.

Vahrenhold, J., Nardelli, E., Pereira, C., Berry, G., Caspersen, M. E., Gal-Ezer, J., Kölling, M., McGettrick, A., & Westermeier, M. (2017). Informatics Education in Europe: Are We All In The Same Boat.

Van der Laan, Krikke, H., Kievit, D., & Bosschaart, E. (2001). Fundament Informatica (2 ed.). Instruct.

Vogel, S., Santo, R., & Ching, D. (2017). Visions of computer science education: Unpacking arguments for and projected impacts of CS4All initiatives. Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, 609–614.

Walden, J., Doyle, M., Garns, R., & Hart, Z. (2013). An informatics perspective on computational thinking. Proceedings of the 18th ACM conference on Innovation and technology in computer science education, 4–9. https://doi.org/10.1145/2462476.2483797

Weert, T. V., & Tinsley, D. (1994). Informatics for secondary education: A curriculum for schools. UNESCO.

Weintrop, D., & Wilensky, U. (2013). Robobuilder: A computational thinking game. SIGCSE, 736.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. Journal of Science Education and Technology, 25(1), 127–147.

Weller, M. P., Do, E. Y. L., & Gross, M. D. (2008). Escape machine: Teaching computational thinking with a tangible state machine game. Proceedings of the 7th international conference on Interaction design and children, 282–289.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012). The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. Proceedings of the 43rd ACM technical symposium on Computer Science Education, 215–220. https://doi.org/10.1145/2157136.2157200

Whalley, J., Clear, T., Robbins, P., & Thompson, E. (2011). Salient elements in novice solutions to code writing problems. Proceedings of the Thirteenth Australasian Computing Education Conference-Volume 114, 37–46.

Wilensky, U. (1997). NetLogo Wolf Sheep Predation model. http://www.netlogoweb.org

Wilensky, U. (1999). NetLogo. http://www.netlogoweb.org

Wilensky, U. (2014). Computational thinking through modeling and simulation. Whitepaper presented at the summit on future directions in computer education. Orlando, FL. http://www.stanford.edu/coopers/2013Summit/WilenskyUriNorthwesternREV.pdf.

Wilensky, U., & Papert, S. (2010). Restructurations: Reformulations of knowledge disciplines through new representational forms. Constructionism.

Wilensky, U., & Rand, W. (2015). An introduction to agent-based modeling: Modeling natural, social, and engineered complex systems with NetLogo. MIT Press.

Wilensky, U., Brady, C. E., & Horn, M. S. (2014). Fostering computational literacy in science classrooms. Communications of the ACM, 57(8), 24–28.

Wilson, C. (2010). Running the Empty: Failure to Teach K-12 Computer Science in the Digital Age. Association for Computing Machinery.

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

Wing, J. M. (2008). Computational Thinking and Thinking about Computing. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366(1881), 3717.

Wing, J. M. (2014). Computational thinking benefits society. 40th Anniversary Blog of Social Issues in Computing, 2014, 26.

Wolf, K., & Stevens, E. (2007). The role of rubrics in advancing and assessing student learning. The Journal of Effective Teaching, 7(1), 3–14.

Wolz, U., Stone, M., Pulimood, S. M., & Pearson, K. (2010). Computational thinking via interactive journalism in middle school. Proceedings of the 41st ACM technical symposium on Computer science education, 239–243.

Yadav, A., & Berges, M. (2019). Computer Science Pedagogical Content Knowledge: Characterizing Teacher Performance. ACM Transactions on Computing Education (TOCE), 19(3), 29.

Yadav, A., Berges, M., Sands, P., & Good, J. (2016). Measuring computer science pedagogical content knowledge: An exploratory analysis of teaching vignettes to measure teacher knowledge. Proceedings of the 11th Workshop in Primary and Secondary Computing Education, 92–95.

Yadav, A., Gretter, S., Good, J., & McLean, T. (2017). Computational thinking in teacher education (pp. 205–220). Springer.

Yadav, A., Mayfield, C., Zhou, N., Hambrusch, S., & Korb, J. T. (2014). Computational thinking in elementary and secondary teacher education. ACM Transactions on Computing Education (TOCE), 14(1), 5.

Yadav, A., Zhou, N., Mayfield, C., Hambrusch, S., & Korb, J. T. (2011). Introducing computational thinking in education courses. Proceedings of the 42nd ACM technical symposium on Computer science education, 465–470.

Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An exploration of three-dimensional integrated assessment for computational thinking. Journal of Educational Computing Research, 53(4), 562–590.

Zwaneveld, B., van Dijk, B., Timmers, J., & Heil, I. (2007). Acht jaar CODI: Van omscholen naar opleiden (Eight Years of CODI: From Professional Retraining to a Regular Teacher Training Program) (SSRN Scholarly Paper ID 2947252). Social Science Research Network. https://papers.ssrn.com/abstract=2947252

Appendices

Appendix A: 2007 Dutch Secondary CS Curriculum

2007 Dutch Secondary CS curriculum

The exam:

The exam takes the form of an individual school exam.

The curriculum consists of the following themes:

Theme A Computer science in perspective
Theme B Terminology and skills
Theme C Systems and their structures
Theme D Usage in a context

The school exam:

The school exam is associated with Themes A through D; and in cases where the authorities so decide, it can also include other subject matter that does not necessarily need to be identical for all students.

The subject matter:

Theme A: Computer science in perspective

Sub-theme A1: Science and Technology

1. The student should be familiar with the history of Computer science and IT, their current use and the prospects for future development.

Sub-theme A2: Society

2. The student should be familiar with the role computer sciece and IT play in the developments taking place in society, both in the past and the present.

Sub-theme A3: Study and Career

3. The student should be familiar with the specific functions and jobs performed by computer sciece and IT specialists, and with the role computer sciece and IT play in vocational education and occupations in general. The student should be able to assess to what degree his own abilities and interests correspond with these.

Sub-theme A4: The Individual

4. The student should master the professional work methods that computer science and ICT specialists use, especially those concerning working on a project basis. He should be familiar with the moral values involved in the use of computer science and IT.

Theme B: Terminology and skills

Sub-theme B1: Data representation in a computer

5. The student should be able to describe and use common digital data encoding. Sub-theme B2: Hardware
6. The student should be familiar with the operational functions of a computer, its hardware and common peripheral devices, and should be able to describe the relationships among these functions.

Sub-theme B3: Software

7. The student should be familiar with simple data types, program structures and programming techniques.

Sub-theme B4: Organizations

8. The student should have a global picture of how businesses are structured. He should be familiar with the characteristics of project organization and should be able to explain why a particular type of organization is chosen when a company's information system undergoes major modifications.

Theme C: Systems and their structures

Sub-theme C1: Communication and Networks

9. The student should be familiar with the topological structure and communication layers of a network, and their characteristics. He should also be capable of describing a simple communications protocol, differentiating between its elements and describing them. Furthermore, he should be aware of the security aspects of the Internet.

Sub-theme C2: Operating Systems

10. The student should be familiar with the basic functions of the most common operating systems pertaining to the management of CPU time, memory, data storage media, peripheral devices, and access rights.

Sub-theme C3: Systems in Practice

11. The student should be familiar with the characteristics of, and distinction among real-time systems, expert systems, simulation systems and embedded systems.Sub-theme C4: Development of Information Systems
12. The student should have a global picture of system development stages, and their related actions and products.

Sub-theme C5: Information Flow

13. The student should be capable of describing information flow in a small (business) organization.

Sub-theme C6: Information Analysis

14. The student should be able to analyze information and information requirements and build/adjust the data model accordingly.

Sub-theme C7: Relational Databases

15. The student should be able to name the elements of a relational scheme and describe their meaning. He should be able to translate an information question into a relational database query language command.

He should be familiar with the characteristics and aspects of data management systems, and he should be able to name them and use them for specific systems. (Only pre-university education)

Sub-theme C8: Human-Computer Interaction

16. The student should be able to identify the human-computer interaction element in information systems. He should be familiar with its characteristics and he should be able to recognize and utilize key criteria in the development of user dialogs.

Sub-theme C9: System Development Lifecycle

17. In a simple system development lifecycle, the student should be capable of assessing its progress, testing a prototype, checking whether the final product meets the client's specifications, and assessing whether the system complies with the requirements and wishes of the end user.

Theme D: Usage in a context

18. The student should be familiar with the methods and procedures of project management, as well as the project aspects of system development (Schmidt, 2006).

Appendix B: 2019 Dutch Secondary CS Curriculum

2019 Dutch Secondary CS Curriculum
Computer Science Curriculum havo/vwo

The final exam
The final exam consists of the school exam.

The school exam
The school exam covers:

- The entire domain A, combined with:
- Domains B through F;
- (**havo:**) a selection of one of the domains G through N, and a selection of one of the domains O through R; either the competent authority can make this selection, or the selection is left to the candidate;
- (**vwo:**) a selection of four of the domains G through R, with at least one from the domains G through N, and one from the domains O through R; either the competent authority can make this selection, or the selection is left to the candidate;
- other subject matter may be added if the competent authority chooses to do so; this can vary per candidate.

The Learning Objectives

Core program

Domain A: Skills
General skills
Sub-domain A1: Using information skills

1. The candidate is able to search for, assess, select and process relevant information.

Sub-domain A2: Communicating

2. The candidate is able to communicate adequately, in writing, orally and digitally, in the public domain about topics related to computer science.

Sub-domain A3: Reflecting on learning

3. The candidate is able to reflect on his/her own interests, motivation and learning process when acquiring subject knowledge and skills.

Sub-domain A4: Orientation on study and profession

4. The candidate is able to indicate how the knowledge of computer science is applicable in study and profession, and is able, partly on the basis of this, to voice his/her own interest in related studies and professions.

Scientific skills

Sub-domain A5: Researching

5. The candidate is able

(vwo:) to analyse research questions in contexts, using relevant terminology and theory, translating these into a subject-specific research, carrying out that research, and using the research results to draw conclusions. The candidate is able to use consistent reasoning.

(havo:) to perform instructions for research in contexts, based on research questions, and to draw conclusions from the research results. The candidate is able to use consistent reasoning.

Sub-domain A6: Modelling

6. The candidate is able to use context to analyse a relevant problem, limit this to a manageable problem, translate this into a model, generate and interpret model results, and test and assess the model The candidate is able to use consistent reasoning.

Sub-domain A7: Appreciating and deciding

7. The candidate is able, in contexts, to offer a substantiated decision about a practical situation or a technical application, and is able to distinguish between scientific arguments, normative social considerations and personal views.

Computer science-specific skills

Sub-domain A8: Designing and developing

8. The candidate is able, in a context, to see options to use digital artefacts, translate these options into an objective for the design and development, taking into account technical factors, environmental factors and human factors, to specify desires and requirements, and to test the attainability of these elements, to design a digital artefact, to weigh options for the design of a digital artefact through research and experiments, to implement a digital artefact, and to evaluate the quality of digital artefacts, and to combine these skills for the development of digital artefacts.

Sub-domain A9: Computer science as a perspective

9. The candidate is able to identify phenomena in contexts, explain and interpret these in terms of computer science, recognise and link computer science concepts, and to estimate and argue the possibilities and limitations of digital artefacts in subject-related terms.

Sub-domain A10: Cooperation and interdisciplinarity

10. The candidate is able to cooperate structurally with a team for the design and development of digital artefacts, and is able to cooperate with the people from the application area.

Sub-domain A11: Ethical conduct

11. The candidate is able to describe the ethical norms and values that play a role in the use and development of digital artefacts; he/she is able to explicitly compare his/her own behaviour with the ethical guidelines and **(vwo:)** to critically analyse his/her own conduct and relate this to ethical dilemmas.

Sub-domain A12: Using the computer science tool set

12. The candidate is able to use the relevant tools for computer science, taking into account any risks and security; these tools include (computer) equipment, operating systems, applications, subject matter terminology, subject conventions and formalisms.

Sub-domain A13: Working in contexts

13. The candidate is able to use the skills from domain A and the concepts from domains B through F, and the optional domains G through R, at least in professional contexts, in social contexts and **(vwo:)** in scientific contexts.

Domain B: Basics

Sub-domain B1: Algorithms

14. The candidate is able to develop a solution for a problem into an algorithm, recognising and using standard algorithms, and investigating the correctness and efficiency of digital artefacts through the underlying algorithms.

Sub-domain B2: Data structures

15. The candidate is able to compare the elegance and efficiency of different abstract data structures.

Sub-domain B3: Machines

16. The candidate is able to use finite machines for the characterisation of certain algorithms.

Sub-domain B4: Grammars

17. The candidate is able to use grammars as tools for the description of languages.

Domain C: Information

Sub-domain C1: Objectives

18. The candidate is able to distinguish objectives for information and data processing, such as *searching* and *processing*.

Sub-domain C2: Identifying

19. The candidate is able to identify information and data in contexts, taking into account the objective.

Sub-domain C3: Representing

20. The candidate is able to represent data in a suitable data structure, keeping the objective in mind; he/she is able to compare the elegance, efficiency and implementability of several representations.

Sub-domain C4: Standard representations

21. The candidate is able to use standard representations of numerical data and media, and is able to relate these to each other.

Sub-domain C5: Structured data

22. The candidate is able to translate a need for information into a search request for a collection of structured data.

Domain D: Programming

Sub-domain D1: Developing

23. The candidate is able, for a given objective, to develop programme components in an imperative programming language, using programming language constructions that support abstractions, and structuring programme components in such a manner that they can be easily understood and evaluated by others.

Sub-domain D2: Inspecting and adapting

24. The candidate is able to explain the structure and functioning of certain programme components, and adapt such programme components based on evaluation or changed requirements.

Domain E: Architecture

Sub-domain E1: Decomposition

25. The candidate is able to explain the structure and functioning of digital artefacts through architectural elements, i.e. in terms of the *physical, logical and application layer levels*, and in terms of the components in these layers, with their interaction.

Sub-domain E2: Security

26. The candidate is able to name some security threats and common technical measures, and relate these to architectural elements.

Domain F: Interaction

Sub-domain F1: Usability

27. The candidate is able to evaluate user interfaces of digital artefacts based on heuristics, and to apply the rules of thumb for *good design* for interfaces to the design and development of digital artefacts.

Sub-domain F2: Social aspects

28. The candidate is able to recognise the impact of digital artefacts on the social interaction and personal privacy, and is able to place these in a historical perspective.

Sub-domain F3: Privacy

29. The candidate is able to reason about the consequences of the changing possibilities of digital artefacts for personal freedom.

Sub-domain F4: Security

30. The candidate is able to name some security threats and common socio-technical measures, and relate these to social and human factors.

Optional themes

Domain G: Optional theme Algorithmic, calculability and logic

Sub-domain G1: Algorithm complexity

31. The candidate is able

(**havo:**) to compare the complexity of certain algorithms, and to recognise and name classical *difficult* problems.

(**vwo:**) to explain the difference between exponential and polynomial complexity; to distinguish algorithms based on this difference, and to recognise and name classical *difficult* problems.

Sub-domain G2: Calculability

32. The candidate is able to characterise and relate calculations on different abstraction levels, and to recognise and name classical incalculable problems.

Sub-domain G3: Logic

33. The candidate is able to express characteristics of digital artefacts in logical formulas.

Domain H: Optional theme Databases

Sub-domain H1: Information modelling

34. The candidate is able to draw up an information model for a simple practical situation and is able to define a database based on this situation.

Sub-domain H2: Database paradigms

35. Apart from the rational paradigm, the candidate is able to describe at least one other database paradigm, and is able to weigh the suitability of the relevant paradigms for a concrete application.

Sub-domain H3: Linked data

36. The candidate is able to link data from different databases (data sources) in an application.

Domain I: Optional theme Cognitive computing

Sub-domain I1: Intelligent behaviour

37. The candidate is able to describe the processes that are needed for *intelligent* behaviour, and is able to analyse how these processes can be used in computer science for the development of digital artefacts.

Sub-domain I2: Optional theme cognitive computing

38. The candidate is able to explain the main characteristics of cognitive computing systems, and to indicate the difference with traditional digital artefacts, and is able to indicate whether the solution of a certain problem is suited for a cognitive computing approach.

Sub-domain I3: Application of cognitive computing

39. The candidate is able to realise a simple application by applying one or more methods and technologies from the field of cognitive computing.

Domain J: Optional theme Programming paradigms

Sub-domain J1: Alternative programming paradigm

40. The candidate is able to describe the characteristics of at least one additional programming paradigm, and is able to develop and evaluate programmes according to that paradigm.

Sub-domain J2: Selecting a programming paradigm

41. The candidate is able to make a comparative paradigm assessment for the solution of a certain problem.

Domain K: Optional theme Computer architecture

Sub-domain K1: Boolean algebra

42. The candidate is able to calculate formulas in Boolean algebra.

Sub-domain K2: Digital circuits

43. The candidate is able to construct simple digital circuits at bit level.

Sub-domain K3: Machine language

44. The candidate is able to write a simple programme in machine language, based on the description of an instruction set architecture.

Sub-domain K4: Variation in computer architecture

45. The candidate is able to explain variations in computer architecture in terms of technological developments and application domains.

Domain L: Optional theme Networks

Sub-domain L1: Network communication

46. The candidate is able to describe and analyse the way the network components communicate with each other, and is able to recognise the scaling impact for communication, offer examples of this and explain the consequences.

Sub-domain L2: Internet

47. The candidate is able to explain the basic principles of the Internet as a network, and is able to indicate the consequences of this network for applications and users.

Sub-domain L3: Distribution

48. The candidate is able to describe the different forms of cooperation and the distribution of functions and data in networks.

Sub-domain L4: Network security

49. The candidate is able to analyse the risks of violation of distributed functions and data, and is able to recommend measures that can prevent this violation.

Domain M: Optional theme Physical computing

Sub-domain M1: Sensors and actuators

50. The candidate is able to recognise and give a functional description of the sensors and actuators that are used by a computer system to perceive and manage the physical environment.

Sub-domain M2: Development of physical computing components

51. The candidate is able to model physical systems and processes in order to establish *real time* steering aspects, and is able to use these models, sensors and actuators to develop a computer system to guard and manage physical systems and processes.

Domain N: Optional theme Security

Sub-domain N1: Risk analysis

52. The candidate is able to analyse risks, threats and vulnerabilities in an ICT application, and is able to focus the analysis on both technical and human factors.

Sub-domain N2: Measures

53. The candidate is able to explain the selection of certain technical and organisational measures to improve security.

Domain O: Optional theme Usability

Sub-domain O1: User interfaces

54. The candidate is able to describe and explain the functioning of user interfaces on the basis of cognitive and biological models.

Sub-domain O2: User research

55. The candidate is able to use user research to evaluate the user interfaces of digital artefacts.

Sub-domain O3: Design

56. The candidate is able to design elements of a user interface.

Domain P: Optional theme User Experience

Sub-domain P1: Analysis

57. The candidate is able to explain the relationship between the design selections of an interactive digital artefact and the expected cognitive, behavioural and affective changes or experiences.

Sub-domain P2: Design

58. The candidate is able to create a graphic design of the user interaction of a digital artefact, justify the design decisions, and implement the user interaction for a simple application.

Domain Q: Optional theme Social and individual influence of computer science

Sub-domain Q1: Social influence

59. The candidate is able to explain and predict the positive and negative effects of computer science and the networking society on the lives of individuals and on society.

Sub-domain Q2: Legal aspects

60. The candidate is able to analyse the legal aspects of the application of computer science in society.

Sub-domain Q3: Privacy

61. The candidate is able to investigate the effects of technical, legal and social measures for privacy-related issues.

Sub-domain Q4: Culture

62. The candidate is able to reason about the influence of computer science on cultural expressions.

Domain R: Optional theme Computational Science

Sub-domain R1: Modelling

63. The candidate is able to model aspects of a different scientific discipline in computational terms.

Sub-domain R2: Simulating

64. The candidate is able to construct models and simulations, and use these for the research of phenomena in that other science field.

Abbreviations

ABM	Agent Based Modeling
AI	Artificial intelligence
CODI	Consortium omscholing docenten informatica
CS	Compter science
CS4HS	computer Science for High Schools
CSER	Computer science education research
CSTA	Computer Science Teacher Association
CT	Computational thinking
HAVO	Hoger algemeen voorbereidend onderwijs
IenI	Vakvereniging informatica en digitale geletterdheid
IT/ICT	Information technology/Information and communication technology
KNAW	Koninklijke Nederlandse academie van wetenschappen
PCK	Pedagogical content knowledge
SLO	Stichting leerplanontwikkeling
SOLO	Structure of observed learning outcome
VMBO	Voorbereidend middelbaar beroepsonderwijs
VWO	Voorbereidend wetenschappelijk onderwijs

Curriculum Vitae

Nataša (1966) was born in Zagreb, Croatia into a family with a strong academic attitude. Her mother was an architect, her father was an engineer, and three of her grandparents were teachers or engineers as well. She attended MIOC: this acronym is world-known in Zagreb and stands for Matematičko-informatički obrazovni centar — a magnet high school for math and computer science. Nataša spent the high school senior year as an exchange student at Swartz Creek High School in Swartz Creek, Michigan, USA. When the school got their first Apple IIe computer during her stay there, she helped the CS teacher with writing computer programs in Pascal.

Back to Croatia, she studied math and CS at Zagreb University. After a number of turbulent years with accidents, illness in the family, a war and a distant Dutch boyfriend, she graduated in 1992 and moved to the Netherlands. She first started working as a freelance interpreter for refugees, and soon thereafter as a math teacher at the asylum-seekers center in Zuidlaren. There she decided to become a qualified math teacher and in 1994 graduated from UCLO in Groningen. In those days with high unemployment rate among teachers, it took her a while to find a job, but she was fortunate to be employed at OSG Sevenwolden in Heerenveen that same year.

With the national curriculum revision of 1998, her school decided to introduce the CS course and Nataša applied for a two-year in-service professional development course to become a qualified CS teacher. She graduated in 2000 and started teaching CS. In 2006, she joined the Department for Teacher Education of the Groningen University as lecturer of CS didactics.

In 2012, Nataša won a grant from the Netherlands Organisation for Scientific Research (NWO) (in Dutch: promotiebeurs voor leraren) to engage in the research project resulting in this thesis. During this project, she participated in numerous national and international research and professional conferences and took part in the development of the new curriculum for the elective CS course in Dutch HAVO and VWO. As a spin-off of this project, she has led the development of teaching materials and professional development courses for teachers.

Since 2019, Nataša had quit her high school teaching position, has been working as a researcher on the Computational Thinking in Context project at Radboud University, has joined the Board of Directors of the American Computer Science Teacher Association as International Representative, and recently she has

joined the National Institute for Curriculum Development (in Dutch: Stichting leerplanontwikkeling) as curriculum developer for digital literacy in secondary education.

Nataša is a single mom of two wonderful teenagers.

Links:
https://www.rug.nl/staff/n.grgurina/
https://www.linkedin.com/in/natasagrgurina/

Research Output

Scientific publications

Grgurina, N., & Tolboom, J. (2008). The first decade of informatics in Dutch high schools. Informatics in Education, 7(1), 55-74.

Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., & Suhre, C. (2016). Defining and observing modeling and simulation in informatics. In Proceedings of the International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (pp. 130-141). Springer, Cham.

Grgurina, N., Barendsen, E., Suhre, C., van Veen, K., & Zwaneveld, B. (2017). Investigating informatics teachers' initial pedagogical content knowledge on modeling and simulation. In Proceedings of the International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (pp. 65-76). Springer, Cham.

Grgurina, N., Barendsen, E., Suhre, C., Zwaneveld, B., & Van Veen, K. (2018). Assessment of modeling and simulation in secondary computing science education. In Proceedings of the 13th Workshop in Primary and Secondary Computing Education (pp. 1-10).

Grgurina, N., Tolboom, J., & Barendsen, E. (2018). The second decade of informatics in Dutch secondary education. In Proceedings of the International Conference on Informatics in Schools: Situation, Evolution, and Perspectives (pp. 271-282). Springer, Cham.

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014, June). Computational thinking in K-9 education. In Proceedings of the working group reports of the 2014 on innovation & technology in computer science education conference (pp. 1-29).

Barendsen, E., Mannila, L., Demo, B., Grgurina, N., Izu, C., Mirolo, C., ... & Stupurienė, G. (2015). Concepts in K-9 computer science education. In Proceedings of the 2015 ITiCSE on working group reports (pp. 85-116).

Barendsen, E., Grgurina, N., & Tolboom, J. (2016). A new informatics curriculum for secondary education in the Netherlands. In Proceedings of the International conference on informatics in schools: Situation, evolution, and perspectives (pp. 105-117). Springer, Cham.

Posters and Presentations

Grgurina, N. (2013). Computational thinking in Dutch secondary education. Paper presented at the Informatics in Schools: Local Proceedings of the 6th International Conference ISSEP 2013–Selected Papers, 119.

Grgurina, N., Barendsen, E., Zwaneveld, B., van de Grift, W., & Stoker, I. (2013). Computational thinking skills in Dutch secondary education. Paper presented at the 8th Workshop in Primary and Secondary Computing Education, 31-32.

Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., & Stoker, I. (2014). Computational thinking in het voortgezet onderwijs: Docentkennis onderzocht. Paper presented at the ORD, Groningen.

Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., & Stoker, I. (2014). Computational thinking skills in Dutch secondary education: exploring teacher's perspective. In Proceedings of the 9th workshop in primary and secondary computing education (pp. 124-125).

Grgurina, N., Barendsen, E., Zwaneveld, B., van Veen, K., & Stoker, I. (2014). Computational thinking skills in Dutch secondary education: Exploring pedagogical content knowledge. In Proceedings of the 14th Koli Calling International Conference on Computing Education Research, 173-174.

Grgurina, N., Barendsen, E., Zwaneveld, B., & van Veen, K. (2014). Computational thinking in Dutch secondary education: Teachers' perspective. In M. Thomas, & M. Weigend (Eds.), Informatik und Natur, 6. Münsteraner Workshop zur Schulinformatik (pp. 27-29). Münster: Westfälischen Wilhelms-Universität Münster.

Grgurina, N., & Barendsen, E. (2014). Informatics education at the crossroads: Round table on the Dutch case. Paper presented at the Gülbahar, Y.; Karatas, E.; Adnan, M. (Ed.), ISSEP 2014: 7th International Conference on Informatics in Schools: Situation, Evolution and Perspectives, 22-25 September 2014 Istanbul, Turkey, 137-138.

Grgurina, N., Barendsen, E., van Veen, K., Suhre, C., & Zwaneveld, B. (2015). Exploring students' computational thinking skills in modeling and simulation projects: A pilot study. In Proceedings of the Workshop in Primary and Secondary Computing Education, 65-68.

Grgurina, N., Zwaneveld, B., & Barendsen, E. (2016). Computational thinking in Dutch secondary education: Modeling and simulation. Paper presented at the Informatik Fur Kinder; 7. Munsteraner Workshop Zur Schulinformatik, 81-84.

Grgurina, N., Barendsen, E., Suhre, C., van Veen, K., & Zwaneveld, B. (2018). Assessment of modeling projects in informatics class. Paper presented at the Constructionism 2018: Constructionism, Computational Thinking and Educational Innovation, 570-576.

Professional publications

Tolboom, J., Kruger, J., & Grgurina, N. (2014). Informatica in de bovenbouw havo/vwo: Naar aantrekkelijk en actueel onderwijs in informatica. Enschede: SLO.

Grgurina, N., van der Veen, R., Velthuizen, V. (2020) Agent-based modeling. Domein R: Computational Science. Amersfoort: SLO. https://ieni.github.io/inf2019/themas/r-computational-science

Acknowledgment

Completing a PhD research project like this one is probably the biggest accomplishment of my professional life. While I traveled the entire route myself, there were a lot of people who made this journey possible through their selfless and continuous help and encouragement. I am grateful to all of you. In particular, I want to thank….

First of all, my past and present thesis advisors: Wim — for putting me on the track of this research journey, Bert — for pointing me in the right direction at a crucial junction and continuously keeping an eye on my travels, and Klaas — for making sure the route I took was suitable for all my passengers to follow along. And, most of all, I want to thank Erik for his help and advice along every step of this journey. For all our dates in London, Berlin and Helsinki — to name just a few — and all the museums and restaurants and other places we visited together. Erik, thank you for being a true friend.

I am grateful to Valentina who has been keeping an eye on my progress and regularly opened my eyes to new opportunities. To countless researchers I met along the way who helped me shape my thoughts and ideas. And in particular, to Jos with whom I regularly have interesting discussions and who always manages to point me to all these interesting jobs and gigs and research projects I have been doing since we met during CODI.

I want to thank all the students, teachers and school administrators who were involved in my research — I could not have done it without your cooperation and commitment. At my old high school, OSG Sevenwolden, I want to thank Gerry for approving my partial leave to engage in this research project, Peter for inspiring me on how to combine teaching high school with research; and all my other colleagues, for showing interest and patiently listening to the accounts of my research progress. Most of all, I want to thank Marijke — first my colleague, and always my friend — for standing by me in my darkest hour and helping me to stay afloat.

My colleagues at the Teacher Education department made my journey so much easier — thanks to all of you who discussed research with me, and discussed kids, and houses in Groningen, and gardens, and all other things life brings upon us. To the bèta's for being such support and inspiration: Enno, Alex, Deniz, Lidewij, Gerrit and Martha. Big thanks to my roommate Marjon for making countless cups of tea and always lending an ear. And of course, to Cor, for his endless patience

when discussing details of my research and for helping me to strike just the right tone when writing papers.

There are many people in my life who helped me to succeed with this research project by helping me survive and thrive. I want to thank my neighbors and friends in Lemmer who stood by my side when my husband Hessel suddenly passed away, and Petra and Kristina who helped to keep our household on track in the years thereafter. Rein and Maja, thanks for all the cheese and wine, practical help at crucial moments in my life, and for just being there for me. Kina and Nina, thanks for all the wine, advice on teenage sons and fine times we are having together. Sanja, it's always fine to talk to you and feel the resonance of our ideas — whether it's about the kids, parents, bosses or jobs. Then there is an app group with people with whom I attended elementary school, which feels like a warm blanket: Osnovnjak. Well, Osnovnjak, keep going! I need you! (And, Corinna, please thank Stephen for helping me with the title!) Finally, Hans, I kind of inherited you from Hessel after his death and we have been helping each other to survive ever since — thank you for all your advice, practical help and lazy times by your fireplace.

To my parents Vladimir and Svjetlana I am eternally grateful for the drive to achieve excellence they instilled in me, and all the love and support they had given me for whatever endeavors I had undertaken to do. I'm sure they would have been proud of me for this achievement — as would Hessel, who always believed in me and supported my career choices. Tanja, my little sister, thanks for your patience with me and my kids, and for your yummy cooking. Now that this research project is over, I hope I'll have more time to spend with you in your studio. Finally, Alex and Lucas, my wonderful sons, thank you for putting up with all the turmoil brought about by my work on this project.