COMPUTATIONAL THINKING IN LESMATERIAAL VOOR INFORMATICA

IDZARD STOKER

ONDER BEGELEIDING VAN ERIK BARENDSEN

Onderwijsinstituut voor Informatica en Informatiekunde
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Radboud Universiteit Nijmegen

SAMENVATTING

Computational Thinking, een term van J.M. Wing (2006), omvat een set vaardigheden die nodig zijn om effectief gebruik te maken van een computer. In de huidige studie is onderzocht op welke wijze Computational Thinking voorkomt in het lesmateriaal op het gebied van de informatica. Dit is onderzocht aan de hand van een kwalitatieve analyse van de lesmethode Informatica Actief en het lesmateriaal van de cursus Programmeren voor Science van de Radboud Universiteit. Voor deze analyse is er een methode in de vorm van een codeerschema ontwikkeld, welke gebruikt kan worden in toekomstig onderzoek naar Computational Thinking. Uit dit onderzoek is gebleken dat Computational Thinking reeds voorkomt in lesmateriaal voor informatica, maar dat dit niet op een consistent educatief niveau gebeurt en dat de onderdelen van Computational Thinking niet evenredig voorkomen. Dit onderzoek dient als een nulmeting voor vervolgonderzoek naar Computational Thinking in informatica lesmateriaal en onderwijs. Vervolgonderzoek is nodig om integratie van Computational Thinking in het onderwijs te bevorderen, waarbij het codeerschema uit deze studie als uitgangspunt kan dienen.

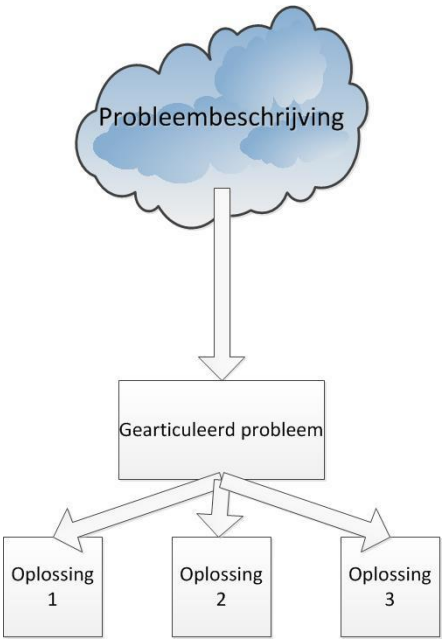
INLEIDING

De digitalisering van informatie en communicatie is doorgedrongen tot alle facetten van de samenleving, een verandering die nog steeds bezig is. Deze ontwikkeling maakt het voor iedereen belangrijk om efficiënt met een computer te kunnen werken en te weten hoe en computer ingezet kan worden om een probleem op te lossen. Wing (2006) onderkent de noodzaak om iedereen deze vaardigheden, welke ze schaart onder de term Computational Thinking (CT), te leren. Volgens Wing is CT een problem solving process dat bestaat uit de verzameling van een aantal mentale processen en gereedschappen die hun oorsprong vinden in de informatica. CT wordt genoemd als een fundamentele vaardigheid en als de geletterdheid van de 21ste eeuw, net zoals taal en rekenen dat was van de 20ste eeuw. Barr en Stephenson (2011) scharen op grond van Wing (2006) de volgende activiteiten onder CT: het zoeken van algoritmische aanpakken bij probleemgebieden, een bereidheid om te wisselen tussen verschillende niveaus van abstractie en representatie, bekendheid met decompositie, ‘separation of concerns’ en modulariteit. Verder dan het beschrijven van CT in de bovenstaande termen en het benadrukken van de noodzaak van CT gaat Wing niet, er wordt geen definitie gegeven.

In Wing (2010) wordt wel een definitie gegeven van CT:

> “*Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent*.” (Wing, Research Notebook: Computational Thinking: What and Why, 2011)

Computational thinking heeft dus betrekking op het herformuleren van een probleem op een zodanige wijze dat het mogelijk wordt deze effectief te laten oplossen door een information-processing agent. Een information-processing agent wordt hierbij breed opgevat en hoeft niet noodzakelijk een computer te zijn, een mens is evengoed een information-processing agent. Barr en Stephenson (Barr & Stephenson, 2011) scharen op grond van Wing’s artikel uit 2006 de volgende activiteiten onder CT: het zoeken van algoritmische aanpakken bij probleemgebieden, een bereidheid om te wisselen tussen verschillende niveaus van abstractie en representatie, bekendheid met decompositie, ‘separation of concerns’ en modulariteit. In figuur 1 wordt een grafisch overzicht gegeven van het problem solving proces waar CT een rol in speelt:



Figuur 1: Problem Solving proces waar CT een rol in speelt

In deze figuur wordt een probleembeschrijving omgezet naar een gearticuleerd probleem, die vervolgens opgelost kan worden door middel van de inzet van information-processing agents. In dit figuur is een gearticuleerd probleem een probleem dat zo geformuleerd is dat het effectief op te lossen is met behulp van information-processing agents. Het bereik van CT in deze figuur in aan discussie onderhevig, maar duidelijk is dat het grootste deel van CT terug komt in de overgang van de probleemstelling naar het gearticuleerde probleem, zoals terug te zien is in de definitie van Wing (2010). Bij deze overgang komt bijvoorbeeld probleem decompositie en abstractie kijken. Daarnaast is het standpunt dat CT ook in de overgang van het gearticuleerde probleem naar de oplossing(en) een rol speelt, ook te verdedigen. Uit de definitie van Wing (2010) blijkt immers dat ook de oplossing(en) op een dergelijke wijze geformuleerd moet worden dat ze op een effectieve manier kunnen worden uigevoerd door een information-processing agent. Een voorbeeld hiervan is automatisering van oplossingen.

De Koninklijke Nederlandse Akademie van Wetenschappen (KNAW) (2012) geeft aan dat CT ook een plaats verdient in het Nederlandse informatica onderwijs op het voortgezet onderwijs (VO). Het huidige onderwijs bereidt de leerlingen niet afdoende voor op de eisen op het gebied van de informatica die de maatschappij aan ze zal stellen wanneer ze klaar zijn met hun opleiding. Om aan deze eisen te voldoen stelt het KNAW een herziening voor van het informatica onderwijs op het VO, waarbij de term *digitale geletterdheid* centraal staat. Onder digitale geletterdheid schaart het KNAW drie onderdelen: basiskennis, gedrag en gebruik. CT valt onder de basiskennis, een onderdeel dat volgens het KNAW in de onderbouw al moeten worden aangeleerd en daarmee aan elke leerling. De vraag rijst nu, hoe kunnen digitale geletterdheid en CT worden geïntegreerd in het informatica onderwijs?

Nataša Grgurina, van de Rijksuniversiteit Groningen is op het moment bezig met het uitvoeren van een promotieonderzoek dat aansluit op het adviesrapport van het KNAW, namelijk naar de integratie van CT in het informatica onderwijs in het VO. Dit onderzoek is nog in een initiële fase, maar zou van

veel waarde kunnen zijn voor de ontwerpers van nieuw informatica lesmateriaal. De eerste stap in dit onderzoek is bekijken hoe CT voorkomt in huidig lesmateriaal voor informatica. Met die kennis kan de huidige staat van het informatica lesmateriaal op het gebied van CT vastgesteld worden en kan gekeken worden waar verbeterpunten zitten op het gebied van CT.

Dit onderzoek levert hier een bijdrage aan door lesmateriaal voor informatica te onderzoeken met als hoofdvraag:

'Op welke wijze komt Computational Thinking voor in lesmateriaal op het gebied van informatica?'.

Deze hoofdvraag wordt beantwoord aan de hand van de volgende drie deelvragen:

1. *Op welke wijze kan bepaald worden welke aspecten van CT in bronnen voorkomen?*
2. *Welke inhoudelijke aspecten van CT komen voor in het lesmateriaal?*
3. *Welke variatie is te herkennen tussen de verschillende lesmaterialen ten opzichte van educatieve niveaus?*

Waarbij bij de tweede deelvraag ook in zal worden gegaan op de inhoudelijke variatie binnen elk aspect. Voor het bepalen van het educatieve niveau uit de derde deelvraag zal gebruik gemaakt worden van de taxonomie van Bloom, welke verder zal worden uitgelegd in het theoretisch kader.

In dit onderzoek worden twee lesmaterialen onderzocht. Ten eerste zal er een lesmethode uit het voortgezet onderwijs worden onderzocht. Daarnaast zal er lesmateriaal worden onderzocht dat voortkomt uit het wetenschappelijk onderwijs, namelijk van de Radboud Universiteit Nijmegen. Het verwachte resultaat van dit onderzoek bestaat uit twee onderdelen. Ten eerste wordt er een methode verwacht die gebruikt kan worden om CT te herkennen in textuele bronnen. Deze methode is gelimiteerd tot textuele bronnen omdat in dit onderzoek enkel deze soort bronnen worden onderzocht en de methode zodoende daarvoor geschikt is. Daarnaast is de verwachting dat het resultaat van de analyse van de teksten erop zal duiden dat de CT aspecten voor komen in de onderzochte lesmaterialen. Dit is de verwachting omdat in het huidige informatica curriculum ook al onderdelen van CT zijn terug te vinden (SLO, 2007, p. 43;44).

Voor het vak informatica in het voortgezet onderwijs bestaan drie lesmethoden: Fundering, Enigma en Informatica Actief (SLO, 2007). Uit de data van het SLO blijkt dat Informatica Actief de meest gebruikte lesmethode is, deze methode wordt namelijk gebruikt op 50 procent van alle middelbare scholen die het vak informatica aanbieden. Omdat Informatica Actief de meest gebruikte lesmethode is, is ervoor gekozen om deze methode te gebruiken voor dit onderzoek.

ateriaal voor het VO wordt er ook door andere partijen lesmateriaal gemaakt op het gebied van informatica. Dergelijk materiaal is zeer divers en een compleet overzicht van al het materiaal bestaat niet. In dit onderzoek is ervoor gekozen om de opdrachten van de cursus Programmeren voor Science (PvS) te analyseren, welke wordt gegeven aan studenten van de opleiding Science op de Radboud Universiteit Nijmegen (RU). Dit materiaal is samengesteld door Sjaak Smetsers, universitair docent aan de RU. Het materiaal van PvS is gemaakt met als doel aan te sluiten op modelleren, probleem decompositie en algoritmen, allemaal aspecten van CT. PvS is gekozen om te onderzoeken door deze overeenkomsten met CT.

THEORETISCH KADER

COMPUTATIONAL THINKING IN HET ONDERWIJS

Aansluitend op het werk van Wing hebben de International Society for Technology in Education (ISTE) en de Computer Science Teacher Association (CSTA) een definitie opgesteld voor CT in het basis- en middelbaar onderwijs in de Verenigde Staten (ISTE; CSTA, 2011). De operationele definitie die het ISTE/CSTA geven is tweeledig:

Ten eerste is CT een problem solving proces dat de volgende karakteristieken heeft (maar niet gelimiteerd is tot deze karakteristieken):

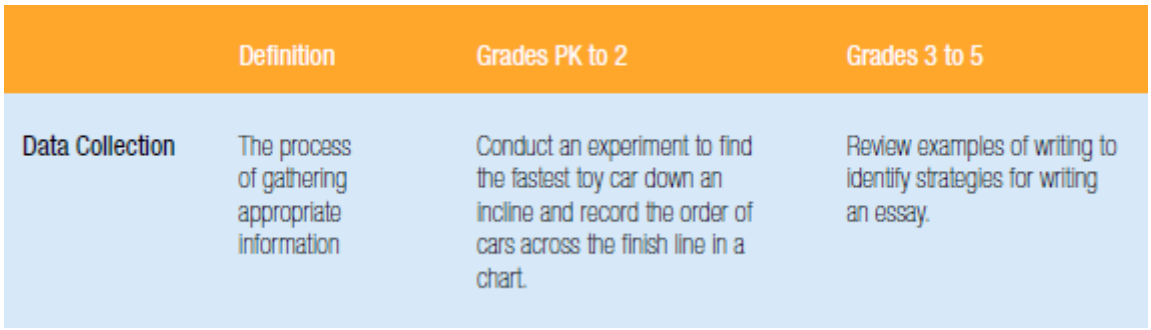
- Het formuleren van problemen op een zodanige manier dat het mogelijk wordt om een computer of ander gereedschap te gebruiken om het probleem op te lossen.
- Het logisch ordenen en analyseren van data.
- Het representeren van data door middel van abstracties zoals modellen en simulaties.
- Het automatiseren van oplossingen door middel van algoritmisch denken
- Het identificeren, analyseren en implementeren van mogelijke oplossingen met als doel de meest effectieve en efficiënte oplossing te vinden.
- Het generaliseren en verplaatsen van dit problem solving proces naar een grote variatie aan problemen.

Daarnaast worden deze karakteristieken aangevuld door een verzameling attitudes die degene die CT beheerst dient te hebben. Deze attitudes behelzen:

- Met vertouwen om kunnen gaan met complexiteit.
- Het hebben van doorzettingsvermogen bij lastige problemen
- Het vermogen te kunnen omgaan met ambiguïteit
- Het vermogen te kunnen omgaan met open problemen
- Het vermogen met anderen te communiceren en te werken om een gezamenlijk doel te bereiken.

Deze definitie wordt door het ISTE/CSTA verder uitgewerkt tot een vocabulaire, die bestaat uit negen onderwerpen die samen alle aspecten van CT bevatten. Bij elk onderwerp wordt een voorbeeld gegeven, waardoor een duidelijk overzicht ontstaat van de verschillende onderdelen en hun invulling. Het vocabulaire wordt vergezeld door een progression chart.

In dit document wordt verder een begrippenlijst, in het document ‘vocabulaire’ genoemd, van CT gegeven waarin een operationele definitie van CT wordt uitgewerkt tot negen onderwerpen. Deze onderwerpen zijn alle aspecten van CT die ook beschreven zijn in de operationele definitie, maar daar nog impliciet waren. Dit vocabulaire is dus de operationele definitie van CT zoals die hierboven is gegeven, maar dan opgedeeld in negen onderwerpen. Naast dit vocabulaire bevat het document nog een ‘progression chart’ die per onderdeel uit het vocabulaire aangeeft welk niveau een leerling op dat onderdeel moet hebben op welke leeftijd. Figuur 2 (ISTE; CSTA, 2011) is een voorbeeld van een deel van het vocabulaire met progression chart, waarbij van het onderdeel *Data collection* de definitie en de progression chart tot klas 5 wordt weergegeven. De hele progression chart geeft voorbeelden tot klas 12.



	Definition	Grades PK to 2	Grades 3 to 5
Data Collection	The process of gathering appropriate information	Conduct an experiment to find the fastest toy car down an incline and record the order of cars across the finish line in a chart.	Review examples of writing to identify strategies for writing an essay.

Figuur 2: Progression Chart van Data Collection tot klas 5

In tabel 1 wordt een overzicht gegeven van de onderdelen van CT die in het vocabulaire worden genoemd met daarbij een definitie en een voorbeeld. De definitie is uit het vocabulaire uit de Teacher Resources gehaald en het voorbeeld komt uit de progression chart van hetzelfde document, zoals hierboven in figuur 2 te zien is.

Aspect	Definitie	Voorbeeld
Data Collection	Proces van het verzamelen van relevante informatie.	Het verzamelen van informatie via experimenten, interviews, enquêtes of literatuurstudie.
Data Analysis	Begrijpen van data, vinden van patronen, trekken van conclusies.	Het evalueren van grafieken of het toepassen van relevante statistische methodes.
Data Representation	Selecteren en organiseren van informatie in relevante grafieken, tabellen, woorden en plaatjes.	Het maken van grafieken van data, het selecteren van de effectiefste representatie uit een verzameling van representaties en het manipuleren van conclusies door middel van het selecteren van een bepaalde vorm van representatie.
Problem Decomposition	Opdelen van een taak in kleinere, behapbare taken.	Het opdelen van een lange lijst met opdrachten in subcategorieën en het plannen van een project door middel van deelprojecten.
Abstraction	Reduceren van complexiteit om algemene concepten over te brengen.	Het vergelijken van twee verschillende concepten op algemene ideeën.
Algorithms & Procedures	Opstellen van een serie van geordende stappen om een probleem op te lossen of een bepaald doel te bereiken.	Het beschrijven van de route van A naar B.
Automation	Saaie en repetitieve taken laten uitvoeren door computers.	Barcodes en kassa's.
Simulation	Representeren of modelleren van een proces, of het uitvoeren van een experiment door middel van modellen.	Het uitvoeren van een routebeschrijving om te controleren of die klopt.
Parallelization	Organiseren van middelen op een dergelijke wijze dat het mogelijk wordt om ze simultaan in te zetten om een gezamenlijk	Het maken van een planning en het toewijzen van taken aan teamleden tijdens een project.

doel te bereiken.

Tabel 1: Aspecten van CT, hun definitie en een voorbeeld

Als laatste worden in de Teacher Resources nog negen Computational Thinking Learning Experiences (CTLE's) weergegeven. Deze CTLE's zijn lesplannen waarin CT is geïntegreerd. De lesplannen zijn bedoeld voor verschillende vakgebieden en voor verschillende leeftijden. Op elke pagina van de lesplannen staat een kolom aan de rechter kant waarin de besproken aspecten van CT worden aangeven.

DEFINITIE

De definities van Wing (2010) (2006) en de definitie van het ISTE/CSTA (2011) overlappen op een groot aantal punten. Beide definiëren CT als een mentaal proces dat hoort bij problem solving. Daarnaast zijn de punten over decompositie, abstractie, representatie en algoritmisch denken uit de beschrijving van Wing (2006) vergelijkbaar met de punten over *automation*, *problem decomposition*, *abstraction*, *data representation* en *algorithms & procedures* uit de definitie van het ISTE/CSTA (2011). Ook in de definitie van Wing (2010) zijn overeenkomsten te vinden met de definitie met ISTE/CSTA. Hier zijn de twee partijen het er voornamelijk over eens dat het beoordelen van de berekenbaarheid van een probleem een karakteristiek is van CT.

Als definitie in dit onderzoek zal de operationele definitie van het ISTE/CSTA (2011) gebruikt worden. Er is voor deze definitie gekozen omdat deze ontwikkeld is voor het onderwijs, duidelijk is afgebakend en explicieter is dan de definitie van Wing. Deze karakteristieken zorgen ervoor dat de definitie beter bruikbaar is om determinanten van CT vast te stellen in dit onderzoek.

BLOOM'S TAXONOMIE

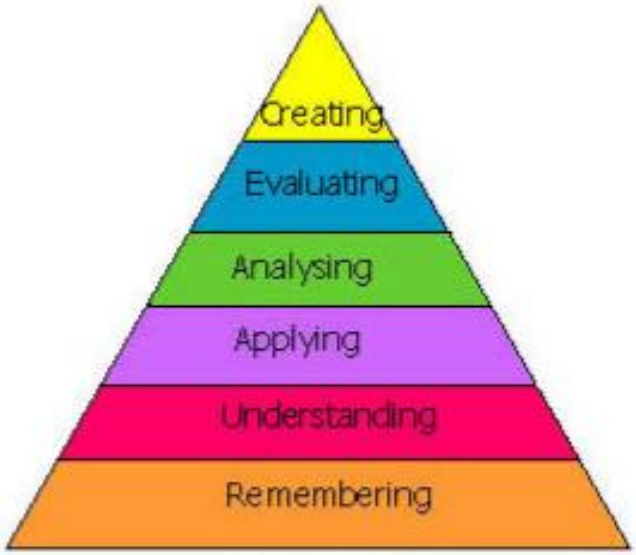
Forehand (2010) beschrijft dat Benjamin S. Bloom een groep docenten heeft geleid in een ambitieuze taak om educatieve doelen te classificeren, met de bedoeling een methode te ontwikkelen om de belangrijke processen voor leren te kunnen classificeren. Dit heeft geleid tot een taxonomie op drie domeinen: het cognitieve domein, het affectieve domein en op het psychomotorische domein. Bloom et al (1956) is het resultaat van acht jaar studie naar dit onderwerp. Voor het onderzoek dat hier besproken wordt, is enkel de taxonomie op het cognitieve domein interessant, dus vanaf nu zal enkel daarover gesproken worden als het gaat over Bloom's taxonomie.

Anderson heeft in 2001 leiding gegeven aan een nieuwe groep onderzoekers die een actualisatie van Bloom's taxonomie hebben uitgevoerd (Anderson, Krathwohl, & Bloom, 2005). In dit onderzoek zal deze geactualiseerde taxonomie gebruikt worden. De taxonomie bestaat uit zes oplopende niveaus: onthouden, begrijpen, toepassen, analyseren, evalueren en creëren. Deze niveaus zijn hiërarchisch, wat wil zeggen dat wanneer een student het tweede niveau beheerst, deze ook het eerste niveau beheerst. De niveaus worden als volgt gekarakteriseerd: (Forehand M. , 2010):

- Onthouden: het herkennen en herinneren van kennis uit het lange termijn geheugen.
- Begrijpen: het construeren van betekenis uit berichten doormiddel van interpretatie, classificeren, samenvatten, vergelijken en uitleggen.
- Toepassen: het afhandelen van een procedure door uitvoering of implementatie.
- Analyseren: materiaal in consistente delen opdelen, bepalen hoe de verschillende onderdelen zich tot elkaar of een structuur of doel verhouden.
- Evalueren: Beslissingen maken gebaseerd op criteria en standaarden doormiddel van nakijken en kritiek leveren

- Creëren: Elementen samenvoegen om een coherent en functioneel geheel te vormen; elementen reorganiseren in een nieuw patroon door middel van genereren, plannen of produceren.

In figuur 3 wordt de taxonomie grafisch weergegeven (Anderson, Krathwohl, & Bloom, 2005). Deze taxonomie wordt in dit onderzoek gebruikt om de gevonden aspecten van CT in de lesmaterialen te classificeren op hun niveau en doel.



Figuur 3: Bloom's taxonomie in het cognitieve domein, geactualiseerd door Anderson

METHODE

In dit onderzoek is er gebruik gemaakt van content analysis. Content analysis komt voor in twee smaken: kwalitatief en kwantitatief. Kwalitatieve content analysis is een onderzoeksmethode waarmee teksten op een empirische, methodische en gecontroleerde manier geanalyseerd kunnen worden, waarbij gelet wordt op hun context in de communicatie (Mayring, 2000). Kwantitatieve content analysis is een onderzoeksmethode waarbij de inhoud van teksten kan worden gekwantificeerd doormiddel van vooraf bepaalde categorieën, op een systematische en repliceerbare manier (Bryman, 2008) (p. 275). In dit onderzoek is gebruik gemaakt van kwalitatieve content analysis, omdat er gekeken wordt op welke wijze CT terug komt in het lesmateriaal. Dit is bereikt door te kijken naar de fragmenten in het lesmateriaal dat aspecten van CT bevat.

Het onderzoek is opgebouwd uit drie fasen. In de eerste fase is er een codeerschema ontwikkeld dat gebruikt is om aspecten van CT te herkennen in de onderzochte documenten. Met de resultaten van deze fase kon er een antwoord gegeven worden op de eerste deelvraag. Het codeerschema is tot stand gekomen door middel van een iteratief proces van drie iteraties. In de eerste iteratie zijn er algemene categorieën gekozen die later gebruikt zijn om de codes bij in te delen. Voor deze categorieën zijn de negen aspecten uit het vocabulaire van de Teacher Resources van het ISTE/CSTA gebruikt (ISTE; CSTA, 2011). De negen categorieën zijn *Data collection, Data analysis, Data representation, Problem decomposition, Abstraction, Algorithms & procedures, Automation* en *Simulation*. In tabel 1 in de sectie 'Theoretisch kader' staan deze negen categorieën genoemd met een definitie en een voorbeeld. Deze negen categorieën vormden het initiële codeerschema, welke nog niet toepasbaar was doordat de onderdelen te algemeen waren.

In de tweede iteratie werd het schema verfijnd door gebruik te maken van de Computational Thinking Learning Experiences (CTLE's) die het ISTE/CSTA aanbiedt in hun Teacher Resources. Dit zijn in essentie lesplannen waarbij CT onderdeel is gemaakt van de les. In deze CTLE's zijn sommige fragmenten in de tekst gemarkeerd met een aspect van CT. Bij elke markering wordt in een kolom aan de rechter kant van de pagina uitleg gegeven over waarom dit fragment een voorbeeld is van CT. Wat het ISTE/CSTA hier dus in essentie heeft gedaan is het coderen van hun eigen lesplannen op de aspecten van CT, welke dezelfde zijn als de categorieën in het initiële codeerschema. In het onderzoek zijn de fragmenten die het ISTE/CSTA heeft gemarkeerd met aspecten van CT gegroepeerd op hun aspect. Vervolgens zijn per aspect de fragmenten gegeneraliseerd tot een verzameling van deelaspecten (deelcodes). Deze deelcodes zijn vervolgens per categorie in het codeerschema gezet. Op deze manier is er een verfijning van het initiële codeerschema tot stand gekomen.

De derde iteratie heeft dienst gedaan als controle van de compleetheid van het codeerschema. In deze iteratie zijn er 93 documenten gecodeerd met behulp van het codeerschema dat is ontstaan in de tweede iteratie. Hierbij is gekeken welke fragmenten uit de teksten door het schema gecodeerd werden en niet. Vervolgens is in de fragmenten die niet gecodeerd waren gekeken of daar nog fragmenten tussen zaten die volgens de definitie onder CT vielen. Al deze CT fragmenten die niet door het schema zijn opgepikt, werden vervolgens aan dezelfde methode onderworpen als in de tweede iteratie is gebruikt. De fragmenten zijn gemarkeerd met een aspect van CT, vervolgens is per aspect op zoek gegaan naar overeenkomsten tussen de fragmenten en zijn er nieuwe deelcodes ontstaan uit de aspecten die onderlinge overeenkomsten hadden. Deze nieuwe deelcodes zijn daarop toegevoegd aan het codeerschema uit de tweede iteratie. Na deze derde iteratie was het codeerschema klaar voor gebruik.

In de tweede fase van het onderzoek is het codeerschema uit de eerste fase gebruikt om de teksten te analyseren op de aspecten van CT. Dit is gedaan door de teksten te coderen met behulp van het software programma Atlas.ti. In totaal zijn er 93 documenten gecodeerd, waarvan er 84 voortkomen uit het materiaal van IA en 9 uit het materiaal van PvS. Omdat het in het tijdsbestek van dit onderzoek niet mogelijk was om al het materiaal van IA te onderzoeken is er besloten een selectie te maken uit dit materiaal. Deze selectie is gemaakt op basis van de overeenkomst van het materiaal met CT, zodat de kans het grootst was om CT aspecten te vinden in het geselecteerde materiaal. Voorbeelden van dergelijke overeenkomsten zijn de hoofdstukken Model Checking en *Simulation* en Hoofdstuk 7 en *Algorithms & procedures*. De resultaten uit deze fase, namelijk de gevonden fragmenten die CT aspecten vertegenwoordigen, zijn vervolgens gebruikt in de volgende fase.

De volgende documenten zijn gecodeerd en zijn daarmee de onderzoekseenheden:

Hoofdstuk	Paragraaf
7: Algoritmen en programma's	1.1; 1.2; 2.1 - 2.3; 3.1; 4.1; 5.1; 7.1
8: Programmeertalen	1.1; 2.1 - 2.3; 3.1 - 3.4; 4.1; 5.1; 6.1
10: Relationele Databases	1.1; 1.2; 2.1; 3.1 - 3.3; 4.1; 4.2; 5.1 - 5.4; 6.1; 6.2; 7.1 - 7.3; 8.1 - 8.3
12: Informatiemodellering	1.1; 1.2; 2.1 - 2.3; 3.1; 4.1 - 4.4; 5.1; 5.2; 6.1 - 6.6; 7.1
13: Organisatie, projecten en projectmatig werken	6.1
Model Checking	1.1 - 1.5; 2.1 - 2.7; 3.1; 3.2; 4.1 - 4.4
Programmeren voor Science	Opdracht 1 t/m 9

Tabel 2: Overzicht van de gecodeerde teksten

In de derde fase zijn de gecodeerde fragmenten geanalyseerd op de verschillende aspecten van CT die ze vertegenwoordigen, waarbij gelet is op de herkomst van de fragmenten. Daarnaast is er per aspect gekeken wat de inhoudelijke variatie is tussen de fragmenten. Met de gegevens uit deze analyse kon de tweede deelvraag beantwoordt worden. Als laatste is er gekeken naar het educatieve niveau van de fragmenten aan de hand van Bloom's taxonomie. Hierbij is een vergelijking gemaakt tussen de niveaus van de fragmenten uit IA en PvS. Aan de hand van de resultaten van deze analyse en vergelijking is een antwoord gegeven op de derde deelvraag.

RESULTATEN

Hieronder wordt per fase van het onderzoek de resultaten getoond. Hierbij wordt de volgorde uit de methode sectie aangehouden.

FASE 1: CODEERSCHEMA

Zoals beschreven staat in de sectie Methode, is er voor het uitvoeren van de tweede iteratie van het ontwerpen van een codeerschema gebruik gemaakt van analyses van de CTLE's van het ISTE/CSTA en bestond de derde iteratie uit een controle op de volledigheid van het codeerschema dat hieruit voort kwam. De resultaten van deze twee iteraties worden hieronder beschreven.

TWEEDE ITERATIE

Zoals aangegeven in de sectie Methode is voor de tweede iteratie gekeken naar de CTLE's in de Teacher Resources van het ISTE/CSTA. In totaal zijn er negen CTLE's die allemaal bekeken zijn. De CTLE's zijn voorbeelden van lessen die leraren zouden kunnen geven aan hun leerlingen. De lessen zijn verdeeld over de verschillende leeftijden van de leerlingen, zodat voor elke leeftijd er een voorbeeld is. Ook zijn de CTLE's gemaakt voor verschillende vakken, waardoor er een brede invulling van CT ontstaat.

De codes en fragmenten die het ISTE/CSTA hebben aangegeven in de CTLE's zijn gegroepeerd op hun categorie en vervolgens is er gekeken wat de overeenkomsten zijn tussen de verschillende fragmenten. In alle CTLE's samen staan er zeven fragmenten over Abstraction, acht over Algorithms & Procedures, zes over Automation, vijf over Data Analysis, vier over Data Collection, drie over Data Representation, een over Paralellization, twee over Problem Decomposition en zeven over Simulation.

Een overzicht van de deelcategorieën staat hieronder, met daarbij een voorbeeld uit de CTLE's.

Code	Deelcode	Voorbeeld
Data Collection	Collecting data	"Have the students [record the stages] of the plant's development." (pagina 20)
Data Analysis	Drawing conclusions	"Looking at the charts and graphs, have students [draw conclusions from the data]." (pagina 20)
	Finding patterns	"Students identify examples of strong and weak persuasive writing and [record commonalities] on a graphic organizer." (pagina 29)
	Making sense of data	"[Students should analyze the list] by applying the criteria for critical issues, and the class should agree on the 5 to 10 issues that match the best." (pagina 45)
Data Representation	Arrange data for analysis	"Students [label and organize the cateogries and create formulas for the cells] in a way that facilitates the calculation of totals for various sub-categories and categories." (pagina 34)
	Organize/represent data	"Keep [charts and graphs] on the grow rate of the various plants." (pagina 20)
Problem Decomposition	Breaking down tasks	"Before they begin, they have to [break up their storyline] into pieces" (pagina 26)
Abstraction	Finding characteristics	"Students [brainstorm the characteristics] a critical issue should have and then refine those characteristics into a set of characteristics that they will look for in all of the issues that they identify." (pagina 44)
	Creating models	"Each group will [create a model] that shows the

		interdependency of the identified variables" (pagina 50)
Algorithms & Procedures	Making sequential steps in a specific order	"Please [give me directions] to get to the front door" (pagina 14)
	Understanding and changing algorithms	"Discuss [what the search engine was "thinking"] when it returned the irrelevant links." (pagina 38)
	Making decisions in algorithms	"The discussion should focus on what the game, and therefore, the author and programmer must anticipate when the player makes a [decision]." (pagina 45)
Automation	Recognizing different forms of automation	"Explore what students know about [sprinkler systems], a computer-based solution." (pagina 18)
	Recognizing the advantages of automation	"[Discuss the advantages that an automated table] has over a simple written list." (pagina 39)
Simulation	Creating pseudo-code	"[Using pseudocode to simulate a computer algorithm] is a useful CT skill throughout the curriculum." (pagina 41)
	Creating models of processes	"Each team of students will [create a flow chart] from a specific critical issue and map where at least one of the paths might lead."(pagina 46)
	Experimenting	"Each group will show and describe how the model will [change if a variable changes]." (pagina 51)
Parallelization	Combine/merge activities	"As a whole-class activity, [combine the alternate paths] from each of the teams." (pagina 46)

Tabel 3: Overzicht van de categorieën en hun deelcodes

In deze tabel is het belangrijk om de verschillen tussen een aantal deelcategorieën te onderscheiden. Zo is er een duidelijk verschil tussen Creating models bij Abstraction en Creating models of processes bij Simulation. De eerste, van Abstraction, gaat specifiek niet over modellen van processen. Daar voorziet de tweede, van Simulation, namelijk in. Instanties over een model van een proces, zoals een flowchart, dienen dus altijd gecodeerd te worden als Creating models of processes, terwijl datamodellen gecodeerd dienen te worden met Creating models.

Daarnaast is er een verschil tussen het maken van een set met stappen op een bepaalde volgorde en het maken van Pseudo-code. Uit de codering van het ISTE/CSTA blijkt dat stappen op een bepaalde volgorde horen bij Algorithms & Procedures en dat pseudo-code bij Simulation hoort. Het idee hierachter is dat pseudo-code een simulatie is van hoe het echte algoritme zich zou gedragen, het is dus een model van het algoritmeproces. Waarom stappen in een bepaalde volgorde geen model is van het algoritmeproces wordt niet duidelijk, maar uit de definitie van de verschillende hoofdcategorieën kan afgeleid worden dat stappen in een bepaalde volgorde duidelijk bij Algorithms & Procedures horen. Over de implementatie van algoritmen wordt in deze CTLE's weinig geschreven, waardoor het niet duidelijk is of een onderdeel is van CT en waar het bij hoort.

DERDE ITERATIE

Op grond van het codeerschema uit de tweede iteratie is er een volledigheidscheck gedaan door een aantal teksten van Informatica Actief en de opdrachten van Programmeren voor Science te coderen met het codeerschema. De volgende teksten zijn hierbij gecodeerd: van Informatica actief H07-9.1, H10-3.1, H10-5.1 en opdracht 2 en 3 van Programmeren voor Science. Uit deze check bleek dat er nog wat codes misten in het schema. De aanvullingen zijn toegevoegd aan het schema en worden hieronder beschreven.

Ten eerste is komen bij het Data Collection niet enkel manieren naar voren voor het verzamelen van data, maar wordt er ook gesproken over het selecteren van relevante data. Dit hoort erbij omdat het bij Data Collection ook belangrijk is om te weten welke data verzameld moet worden, wat ook te halen is uit het stuk "relevante informatie" uit de definitie van Data Collection. Hierdoor is de deelcategorie Selecting relevant data toegevoegd.

Daarnaast hoort bij Algorithms & Procedures ook het implementeren van algoritmen in programmeertalen. Dit komt veel voor in de teksten en is daarnaast af te leiden uit een van de coderingen in de CTLE's van het ISTE/CSTA. Daar wordt namelijk bij de codering Algorithmic thinking op pagina 26 ook over gesproken met de woorden "They begin [creating their animation] of the food chain in Scratch using programming code blocks with commands". Hoewel Wing het in haar stuk uit 2006 op pagina 3 erover heeft dat CT conceptualiseren is en niet programmeren, kan uit haar woorden geconcludeerd worden dat ze het ook niet uitsluit: "Thinking like a computer scientist means more than being able to program a computer". Op grond van deze twee argumenten is ervoor gekozen om Implementing algorithms toe te voegen aan Algorithms & Procedures. Hierbij horen stukken over de implementatie van algoritmen, maar ook stukken programmacode zelf.

Ten slotte dient in de categorie Problem Decomposition een deelcategorie te worden toegevoegd. In de CTLE's wordt namelijk niet gesproken over het samenvoegen van de verschillende deeltaken, terwijl dat in de stukken van Informatica Actief wel naar voren komt. Het gaat hierbij over het samenvoegen van deeltaken die eerder zijn gemaakt om het grote geheel te bereiken. Hiervoor is de deelcategorie Merging subtasks toegevoegd.

Hieronder staan de verschillende toevoegingen in een overzicht met daarbij een voorbeeld.

Categorie	Deelcategorie	Voorbeeld
Data Collection	Selecting relevant data	"We hebben niets aan een combinatieregel waar gegevens van een leerling gecombineerd worden met de uitlening van iemand anders. Het moet wel om dezelfde leerling gaan" (IA-B04H10-8.3)
Algorithms & Procedures	Implementing algorithms	"They begin [creating their animation] of the food chain in Scratch using programming code blocks with commands" (CTLE pagina 26)
Problem Decomposition	Merging subtasks	"Deze twee query's kunnen we combineren tot een enkele query." (IA-B04H10-7.1)

Tabel 4: Aanvullingen op het codeerschema van de tweede iteratie

DEFINITIEF CODEERSCHEMA

Hieronder staat ter volledigheid het volledige definitieve codeerschema.

Code	Deelcode	Voorbeeld
Data Collection	Collecting data	"Have the students [record the stages] of the plant's development." (pagina 20)
	Selecting relevant data	"We hebben niets aan een combinatieregel waar gegevens van een leerling gecombineerd worden met de uitlening van iemand anders. Het moet wel om dezelfde leerling gaan" (IA-B04H10-8.3)
Data Analysis	Drawing conclusions	"Looking at the charts and graphs, have students [draw conclusions from the data]." (pagina 20)
	Finding patterns	"Students identify examples of strong and weak persuasive writing and [record commonalities] on a graphic organizer." (pagina 29)

	Making sense of data	“[Students should analyze the list] by applying the criteria for critical issues, and the class should agree on the 5 to 10 issues that match the best.” (pagina 45)
Data Representation	Arrange data for analysis	“Students [label and organize the cateogries and create formulas for the cells] in a way that facilitates the calculation of totals for various sub-categories and categories.” (pagina 34)
	Organize/represent data	“Keep [charts and graphs] on the grow rate of the various plants.” (pagina 20)
Problem Decomposition	Breaking down tasks	“Before they begin, they have to [break up their storyline] into pieces” (pagina 26)
	Merging subtasks	“Deze twee query's kunnen we combineren tot een enkele query.” (IA-B04H10-7.1)
Abstraction	Finding characteristics	“Students [brainstorm the characteristics] a critical issue should have and then refine those characteristics into a set of characteristics that they will look for in all of the issues that they identify.” (pagina 44)
	Creating models	“Each group will [create a model] that shows the interdependency of the identified variables” (pagina 50)
Algorithms & Procedures	Making sequential steps in a specific order	“Please [give me directions] to get to the front door” (pagina 14)
	Understanding and changing algorithms	“Discuss [what the search engine was “thinking”] when it returned the irrelevant links.” (pagina 38)
	Making decisions in algorithms	“The discussion should focus on what the game, and therefore, the author and programmer must anticipate when the player makes a [decision].” (pagina 45)
	Implementing Algorithms	“They begin [creating their animation] of the food chain in Scratch using programming code blocks with commands” (CTLE pagina 26)
Automation	Recognizing different forms of automation	“Explore what students know about [sprinkler systems], a computer-based solution.” (pagina 18)
	Recognizing the advantages of automation	“[Discuss the advantages that an automated table] has over a simple written list.” (pagina 39)
Simulation	Creating pseudo-code	“[Using pseudocode to simulate a computer algorithm] is a useful CT skill throughout the curriculum.” (pagina 41)
	Creating models of processes	“Each team of students will [create a flow chart] from a specific critical issue and map where at least one of the paths might lead.”(pagina 46)
	Experimenting	“Each group will show and describe how the model will [change if a variable changes].” (pagina 51)
Parallelization	Combine/merge activities	“As a whole-class activity, [combine the alternate paths] from each of the teams.” (pagina 46)

Tabel 5: Het definitieve codeerschema

FASE 2: CODERING

Een globaal overzicht van het resultaat van de codering van de teksten uit Informatica Actief (IA) en Programmeren voor Science (PvS) is te zien in de volgende tabel. In totaal zijn er 94 documenten geanalyseerd, waarin in totaal 234 fragmenten zijn gecodeerd.

Code	IA H07	IA H08	IA H10	IA H12	IA H13	Model Checking	Programmeren voor Science
Abstraction	0	1	1	3	0	0	5
Algorithms & Procedures	18	0	0	0	0	0	69
Automation	0	10	2	1	0	1	1
Data Analysis	1	0	2	24	0	2	3
Data Collection	0	0	5	0	0	0	3
Data Representation	0	0	3	2	0	0	0
Parallelization	0	0	0	0	3	0	0
Problem Decomposition	4	1	4	2	0	2	11
Simulation	1	1	0	0	0	29	19

Tabel 6: Overzicht van de codecategorieën per onderzocht hoofdstuk

In tabel 6 staan per groep teksten het aantal keer dat een bepaalde codecategorie is gevonden. In deze tabel vallen aan aantal dingen op. Ten eerste heeft Programmeren voor Science (PvS) erg veel codes, bijna de helft van het totaal met 111 van de 234 codes. Dit staat in contrast met het aantal woorden dat PvS heeft in vergelijking met Informatica Actief (IA). PvS heeft namelijk 20250 woorden terwijl IA meer dan twee keer zoveel heeft met 43056 woorden. De codedichtheid in PvS ligt dus veel hoger dan die van IA.

Vervolgens is ook te zien dat PvS de hoogste frequentie heeft op een enkele code, namelijk op *Algorithms & Procedures*. Hoofdstuk 7 van IA scoort ook hoog op deze categorie met 18 instanties. Beide hoofdstukken gaan over programmeren. Opvallend is dat hoofdstuk 8 helemaal geen codes op dit gebied heeft, terwijl ook dat hoofdstuk over programmeren gaat. Daarnaast is er in tabel X een duidelijk verschil te zien in de deelcodes die gevonden zijn in hoofdstuk 7 en PvS. In PvS komen voornamelijk de codes *Understanding & changing algorithms* en *Implementing algorithms* voor, terwijl in hoofdstuk 7 voornamelijk het maken van stappen in een bepaalde volgorde terugkomt. Merk op dat deze twee groepen documenten dus voornamelijk complementair zijn en vrijwel niet overlappen op het gebied van codes. Wat daarnaast erg opvalt, is dat in deze groep codes bijna alle codes erg vaak voorkomen, afgezien van de code *Making decisions in algorithms*.

Code	Hoofdstuk 07	Programmeren voor Science
Implementing algorithms	0	23
Making decisions in algorithms	0	2
Making sequential steps in a specific order	17	6
Understanding and changing algorithms	1	38

Tabel 7: Overzicht van de deelcodes van Algorihms & Procedures in Hoofdstuk 7 van IA en PvS

Ten derde valt het op dat Model Checking (MC) en PvS beide hoog scoren op Simulation, terwijl deze hoofdstukken beide over andere onderwerpen gaan. Model Checking gaat over het simuleren van processen om deze modellen te controleren op problemen, terwijl PvS over problemen gaat.

Codes	Deelcodes	Frequentie
Abstraction	Creating models	3
	Finding characteristics	7
Algorithms & Procedures	Implementing algorithms	23
	Making decisions in algorithms	2
	Making sequential steps in a specific order	23
	Understanding and changing algorithms	39
Automation	Recognizing different forms of automation	11
	Recognizing the advantages of automation	4
Data Analysis	Drawing conclusions	4
	Finding patterns	18
	Making sense of data	10
Data Collection	Collecting data	3
	Selecting relevant data	5
Data Representation	Aranging data for analysis	1
	Organizing/representing data	4
Paralellization	Combine/merge activities	3
Problem Decomposition	Breaking down tasks	18
	Merging subtasks	6
Simulation	Creating models of processes	13
	Creating pseudo-code	4
	Experimenting	33

Tabel 8: Frequentie van de codes per deelcode

In tabel 8 staat per deelcode het totaal aantal keer dat die code is voorgekomen. Uit de tabel is af te leiden dat een code gemiddeld ongeveer 11 keer voorkomt, met een standaardafwijking van iets meer dan 10,5. Deze grote standaardafwijking is ook terug te zien in de tabel, aangezien veel van de codes ver boven of ver onder de gemiddelde frequentie zitten. Understanding and changing algorithms komt het vaakst voor met 39 instanties, met Experimenting vlakbij op de tweede plaats.

Wat vooral opvalt aan de tabel is dat de frequentie waarop de verschillende codes voorkomen totaal niet evenwichtig verdeeld is, sommige codes komen heel vaak voor, terwijl anderen niet boven de vijf keer uitkomen. Daarnaast is er ook binnen de categorieën een groot verschil in het aantal keer dat de deelcodes voorkomen. Zo komen drie van de vier deelcodes van Algorithms & Procedures vaak voor, terwijl de laatste, Making decisions in algorithms, maar twee keer voor komt. Uit de data blijkt daarnaast dat verschillende onderdelen van CT, zoals Algorithms & procedures en Data analysis, bovengemiddeld voorkomen in de documenten, terwijl anderen, zoals Data Collection, ondergemiddeld voorkomen.

FASE 3: BLOOM'S TAXONOMIE EN INHOUDELIJKE ANALYSE

Hieronder zal eerst gesproken worden over de analyse van de fragmenten op het gebied van educatieve niveaus op de schaal van Bloom. Vervolgens zullen de gevonden fragmenten per code geanalyseerd worden op inhoudelijke verschillen.

BLOOM'S TAXONOMIE

Als er gekeken wordt naar de verschillen tussen de fragmenten die gevonden zijn in de documenten van IA en de fragmenten die gevonden zijn in PvS valt op dat de fragmenten van PvS hoger scoren op de taxonomie van Bloom dan de fragmenten van IA. In IA zijn voornamelijk fragmenten te vinden die scoren op de eerste drie stappen in de taxonomie van Bloom. Op het eerste niveau zijn er bij IA bijvoorbeeld fragmenten als:

> *"Syntaxfouten herkent het vertaalprogramma altijd. Bij de vertaling en uitvoering van het programma krijg je dan een melding dat er in een bepaalde regel een syntaxfout zit."*(Hoofdstuk 8, paragraaf 4.1)

Hierbij wordt enkel kennis aangeboden die de student zich later kan herinneren. Verder wordt er niet gekeken of de leerling het begrijpt en wordt er ook niet gevraagd om de kennis toe te passen. Dergelijke fragmenten zijn veel te vinden in de documenten van IA. Het volgende niveau van de taxonomie van Bloom, begrijpen, is te zien in de volgende fragmenten uit IA:

> *"De zin gaat zo te zien over een vertrektijd (14.10 uur), maar wat vertrekt er precies? Een trein, een vliegtuig, een geheim agent?"* (Hoofdstuk 12, paragraaf 2.3)

> *"In het model komen nu helemaal geen objecten voor. Als je de bovenste regel bekijkt, staat daar ook onzin. Een voornaam kan helemaal niet in een plaatsnaam wonen!"* (Hoofdstuk 12, paragraaf 4.4)

Hierbij wordt begrip van de student verwacht op het gebied van de lesstof en de gepresenteerde data. Dergelijke fragmenten zijn ook terug te zien in IA, maar zijn al schaarser dan fragmenten op het eerste niveau. Het derde niveau, toepassen, is eigenlijk enkel terug te zien in fragmenten uit het hoofdstuk Model Checking. In dit hoofdstuk komt dit niveau vrijwel in elke paragraaf voor. Een voorbeeld van een fragment op het derde niveau is:

> *"Voeg een nieuwe property in, voer de bovenstaande query in en controleer deze. Het resultaat is nu precies het omgekeerde van het resultaat van de vorige vraag: Property is not satisfied. Er komt immers geen deadlock voor."* (Model Checking, paragraaf 2.6)

Dit fragment behoort niet tot het volgende niveau, analyseren, omdat er niet wordt gevraagd waarom er geen deadlock voorkomt. Doordat deze vragen niet worden gesteld in dit hoofdstuk of de andere hoofdstukken van IA, komen de fragmenten uit IA niet boven het derde niveau in de taxonomie van Bloom.

In PvS komen deze niveaus ook voor, maar komen ook fragmenten voor die op een hoger niveau liggen. Het derde, vierde, vijfde en zesde niveau komen allemaal terug in de teksten van PvS. Het volgende fragment uit opdracht drie is een voorbeeld van het derde niveau, waarin de de student wordt gevraagd kennis toe te passen die hij heeft geleerd:

> *"Teken een top-down schema voor deze taak."* (Opdracht 3)

Het vierde niveau, analyseren, komt ook voor in de opdrachten van PvS. Studenten worden vaak gevraagd om te kijken naar data, bijvoorbeeld een stuk code, en die data dan te analyseren. Het volgende fragment is daar een voorbeeld van:

> *"Probeer dit. Wat zie je? Kun je dit verklaren?"*(Opdracht 3)

De student wordt gevraagd te analyseren wat hij ziet en dat te verklaren. De student bekijkt wat er gebeurt en attribueert dat gedrag vervolgens aan de code die ervoor zorgt dat het gedrag zich voordoet. Het vijfde niveau is terug te zien in het volgende fragment waarin de student wordt gevraagd om een waarde oordeel te vellen over de mogelijkheid om een schema verder op de delen in kleinere onderdelen. Het vellen van een waarde oordeel is een onderdeel van evalueren.

> *"Kun je de taken in het top-down schema nog verder opsplitsen/detailleren in kleinere deeltaken?"* (Opdracht 3)

Het hoogste niveau, creëren, wordt behaald in fragmenten waarin de student wordt gevraagd iets te maken. Deze fragmenten verschillen van de fragmenten die onder het niveau toepassen door te vragen om iets nieuws te creëren vanuit een vage probleembeschrijving. In het niveau toepassen wordt er meer een taak gegeven waarbij het duidelijk is wat er gedaan moet worden. Op het niveau creëren is het niet direct duidelijk wat er gedaan moet worden, waardoor het creatieve proces van de student wordt aangesproken. Het volgende fragment is hier een voorbeeld van:

> *"Implementeer nu het effect van de zwaartekracht op de beweging van een hemellichaam."*(Opdracht 8)

INHOUDELIJKE ANALYSE PER CODE

Algorithms & procedures

Understanding and changing algorithms

Door het hoge aantal voorkomens van deze code is er een verschil te zien tussen de fragmenten die deze code hebben gekregen. Zo gaat een deel van de codes over het voorspellen wat de uitkomst van een algoritme, waarvoor een algemeen begrip van het algoritme nodig is. Daarnaast is er een begrip nodig van hoe de parameters het algoritme beïnvloeden en vice versa. Een van deze fragmenten is de volgende:

> *"Welk antwoord verwacht je als je Fryde zou vragen of er een ei voor haar ligt? Wat gebeurt er als je de methode boolean eggAhead () aanroept?"* (PvS, Opdracht 1, pagina 2)

Diepere kennis van algoritmen wordt gevraagd in andere fragmenten. Hier wordt ingegaan op de details van de werking van algoritmen en minder op de globale output van algoritmes. Het volgende fragment is hier een voorbeeld van:

> *"Beschrijf in jouw eigen woorden wat deze methode doet."* (PvS, Opdracht 2, pagina 4)

Vervolgens zijn er ook fragmenten die vragen om genoeg kennis van een algoritme om dat algoritme te kunnen aanpassen. Voor het aanpassen van een algoritme is ook kennis nodig van de interne werking van het algoritme, omdat het anders niet duidelijk is wat er moet worden aangepast in het algoritme. Aanpassing van een algoritme wordt gevraagd in het volgende fragment:

> *"Change the method populate so that the referee is always placed at those coordinates."* (PvS, Opdracht 6, pagina 6)

Implementing algorithms

Ook de code Implementing algorithms komt vaak genoeg voor om er verschillen te kunnen vinden tussen de gecodeerde fragmenten. Zo zijn er aan de ene kant fragmenten die gaan over het implementeren van een algoritme aan de hand van een duidelijke probleembeschrijving, zoals pseudo-code of een lijst van regels. Het volgende fragment valt daaronder (de taalfout komt ook voor in de tekst):

> *"Now turn you pseudo-code into real code by changing adding the code in the method handleKeyPress."* (PvS, Opdracht 6, pagina 3)"

Aan de andere kant zien we fragmenten die betrekking hebben tot het implementeren van een algoritme aan de hand van een minder eenduidige probleembeschrijving. Het volgende fragment wordt voorgegaan door een paragraaf waarin de voorwaarden waaraan het algoritme moet voldoen globaal staan beschreven. De student wordt vervolgens gevraagd om zelf uit te zoeken hoe dit omgezet moet worden in een algoritme.

> *"Werk dit geheel uit in een methode genaamd void handleBallCollision() die je weer vanuit je act() methode aanroept."* (PvS, Opdracht 7, pagina 3)

Making sequential steps in a specific order

In deze code kunnen we ook twee verschillende soorten fragmenten onderscheiden. Ten eerste zijn er de fragmenten die ingaan op het beschrijven van sequentiële stappen in een bepaalde volgorde door hier voorbeelden van te geven. Dit kan zijn in de vorm van een lijstje stappen of in de vorm van natuurlijke tekst die de stappen uitlegt. Het volgende fragment is een voorbeeld van het laatste geval:

> *"Bekijk het eerste en tweede getal en bepaal de kleinste. Als het eerste getal het kleinst is, vergelijk dan het eerste getal met het derde en bepaal weer de kleinste. Als het tweede getal kleiner is dan het eerste, vergelijk dan het tweede getal met het derde en bepaal welke het kleinst is, daarna..."* (IA, Hoofdstuk 7, Paragraaf 9.1)

Daarnaast zijn er fragmenten die ingaan op de herkenning van stappen binnen algoritmen. Hierbij wordt onder andere gevraagd aan studenten of ze de stappen herkennen van acties die ze zojuist hebben uitgevoerd. Het volgende fragment staat na de instructie om een object 90 graden te laten draaien:

> *"Welke stappen heb je genomen?"* (PvS, Opdracht 2, pagina 1)

Making decisions in algorithms

Deze code komt maar twee maal voor, waardoor er geen verschillen zijn gevonden in de verschillende fragmenten. De fragmenten die bij deze code zijn gevonden hebben betrekking op het vaststellen van de condities waarbij een bepaalde beslissing moet worden gemaakt in een algoritme. Het volgende fragment is daar een voorbeeld van:

> *"When should the game be over? Express your answer in terms of happy monkeys"* (PvS, Opdracht 6, pagina 6)

Simulation

Experimenting

In de fragmenten die zijn gecodeerd onder Experimenting zijn twee verschillende soorten terug te

vinden. Ten eerste zijn er fragmenten die experimenteren beschrijven waarbij het controleren van de volledigheid van een model centraal staat:

> *"Door met het model te experimenteren in de simulator kun je er bijvoorbeeld snel achterkomen dat er iets nog niet klopt ..."* (IA, Model Checking, paragraaf 2.5)

Andere fragmenten gaan in op experimenteren om het gedrag van een model te onderzoeken. Hierbij worden voornamelijk variabelen in het model aangepast om te kijken hoe het model vervolgens reageert. Het volgende fragment beschrijft een dergelijk experiment:

> *"Experimenteer met de eieren op verschillende plekken. Wat doet de methode act() precies? Zorg dat je verschillende situaties probeert, bijvoorbeeld met Fryde aan de rand van de wereld met haar snavel naar buiten gericht, of terwijl ze voor, boven of bovenop een ei staat."*(PvS, Opdracht 1, pagina 4)

Creating models of processes

Hoewel er 13 instanties zijn van deze code, zijn de fragmenten allemaal redelijk gelijk. Allen gaan ze simpelweg over het creëren van modellen van data. Het voornaamste verschil zit in het soort model dat gecreëerd wordt en enkele fragmenten zijn meer toegespitst op het aanpassen van modellen dan op het maken van modellen, maar dit verschil is bij deze fragmenten zo nihil dat ervoor gekozen is om hier geen extra soort fragmenten voor te onderscheiden. Een voorbeeld van een fragment met deze code is de volgende:

> *"Geef deze stappen weer in een stroomdiagram."* (PvS, Opdracht 2, pagina 5)

Creating pseudo-code

Deze code heeft maar vier instanties en ook deze instanties verschillen onderling zo weinig dat hier ook geen sprake is van verschillende soorten fragmenten. Het enige verschil tussen de fragmenten in deze code bestaat eruit dat enkele fragmenten een voorbeeld geven van pseudo-code, terwijl anderen de student vragen om zelf pseudo-code te schrijven. Een voorbeeld van dat laatste is het volgende fragment:

> *"First write the steps down in pseudo-code (so, in English):"* (PvS, Opdracht 6, pagina 3)

Data Analysis

Finding patterns

In de achttien keer dat deze code voorkomt zijn er twee soorten fragmenten te vinden. Ten eerste hebben we de fragmenten die puur gaan over het vinden van patronen in data, niets meer en niets minder.

> *"We hebben nu een patroon gevonden waaraan de vier zinnen voldoen: de zinnen van de soort woonplaats zien er allemaal uit als '<Persoon aangeduid met voornaam> woont in <Plaats aangeduid met plaatsnaam>'. Bij de invulplekken tussen <> staat precies wat er ingevuld mag worden."* (IA, Hoofdstuk 12, paragraaf 2.2)

De tweede soort fragmenten gaat over het vinden van relaties tussen objecten die in patronen voorkomen. Het volgende fragment is hier een voorbeeld van:

> *"Welke andere 'is-een' relaties zie je?"* (PvS, Opdracht 1, pagina 3)

Making sense of data

Ondanks dat deze code tien keer is voorgekomen in de tekst zien we bij deze code geen verschillende soorten fragmenten. Er zijn wel verschillen tussen de fragmenten, maar er zijn geen groepen fragmenten te vinden met onderlinge gemeenschappelijke eigenschappen anders dan dat ze onderdeel zijn van deze code. Een voorbeeld van een fragment is de volgende:

> *"De zin gaat zo te zien over een vertrektijd (14.10 uur), maar wat vertrekt er precies? Een trein, een vliegtuig, een geheim agent? Om de zin te kunnen analyseren moet je weten waar de zin over gaat."* (IA, Hoofdstuk 12, paragraaf 2.3)

Drawing conclusions

Fragmenten die bij deze code hoorden waren lastig te vinden. De fragmenten die gekoppeld zijn aan deze code hebben allen geen duidelijke overeenkomsten en over sommige valt te discussiëren of de het fragment niet ook onder een andere code zou kunnen vallen. Een voorbeeld:

> *"Dit betekent dat bij de zinnen van dit type ieder dvd-nummer maar één keer mag voorkomen, oftewel het dvd-nummer is bij dit feittype uniek."* (IA, Hoofdstuk 12, paragraaf 6.2)

Problem decomposition

Breaking down tasks

Deze code komt 18 keer voor in de documenten, desondanks is ook bij deze code geen sprake van meerdere soorten fragmenten. Het opdelen van taken wordt in de teksten vaak voorgedaan en er wordt soms gevraagd aan de student om het te doen. Een enkele keer komt een stukje uitleg voor over de manieren om taken op te delen in programmeertalen. Het eerst volgende fragment bestaat uit een voorbeeld van het opsplitsen van taken en het tweede fragment is een vraag aan de student om dat te doen:

> *"We splitsen de Fryde's opdracht op in twee deeltaken: het opzoeken van de rechterbovenhoek en het leggen van de eieren langs de grens."* (PvS, Opdracht 3, pagina 2)

> *"Kun je de taken in het top-down schema nog verder opsplitsen/detailleren in kleinere deeltaken?"* (PvS, Opdracht 3, pagina 3)

Merging subtasks

Deze code is een zes keer gevonden in de teksten, maar was die zes keer niet heel duidelijk. Vaak werd in de tekst niet duidelijk aangegeven wanneer, waarom en hoe taken bij elkaar werden gevoegd. Dit onderdeel werd vaak impliciet verondersteld. Een voorbeeld van een fragment waar het samenvoegen wel duidelijk naar voren kwam is de volgende:

> *"Deze twee query's kunnen we combineren tot een enkele query."* (IA, Hoofdstuk 10, paragraaf 7.1)

Abstraction

Finding characteristics

De fragmenten die bij deze code gevonden zijn gaan vooral over het vinden van een algemene beschrijving van methoden. De bedoeling was vaak om het algemene idee van de werking van een methode uit te drukken in een naam voor die methode. Maar ook het vinden van de verschillende eigenschappen van data kwam terug in de tekst, bijvoorbeeld in een paragraaf over informatiemodellering:

"Een boek - een werk dat geschreven is door een schrijver - is iets anders dan een exemplaar van een uitgave van dat boek. Zo kunnen er verschillende exemplaren van een boek zijn."(IA, Hoofdstuk 10, paragraaf 2.1)

Creating models

Het maken van modellen van processen komt maar weinig voor in de teksten. De fragmenten die wel voorkomen gaan niet over hoe een model gemaakt moet worden. Wel wordt er gevraagd aan de studenten om een model te maken, bijvoorbeeld in dit fragment:

"*first draw a top down-scheme on paper for your program;"* (PvS, Opdracht 5, pagina 2)

Automation

Recognizing different forms of automation

In de tekst komen twee verschillende vormen van automatisering voor. Ten eerste is er automatisering van handelingen met fysieke objecten, zoals barcodes. Daarnaast zijn er fragmenten waar automatisering besproken wordt binnen programma's, zoals database management systemen. Hieronder van beide een voorbeeld in de volgorde zoals ze hierboven staan genoemd:

"De methode om programma's in te voeren werd in de loop van de tijd verbeterd, door het gebruik van ponskaarten en nog later toetsenborden." (IA, Hoofdstuk 8, paragraaf 2.1)

"De computer kan het zelf omzetten in machinetaal-instructies met behulp van het vertaalprogramma."(IA, Hoofdstuk 8, paragraaf 2.1)

Recognizing the advantages of automation

Fragmenten met deze code waren spaarzaam. De fragmenten die voorkwamen waren voornamelijk teksten waarin voorbeelden werden gegeven van automatisering met een uitleg waarom dat voordelig was, bijvoorbeeld een tekst over het automatisch vertalen van het hexadecimale stelsel naar machinetaal, waardoor het invoeren van programma's minder werk was en er minder fouten werden gemaakt. Het volgende fragment is een voorbeeld waarbij de student gevraagd werd wat de voordelen waren van automatisering:

"Welk voordeel heeft het om dit op deze wijze te definieren in plaats van in de code telkens een getal te gebruiken?" (PvS, Opdracht 3, pagina 4)

Data collection

Selecting relevant data

De fragmenten die bij deze code zijn gevonden gingen onder andere over het selecteren van de data die nodig was om een vraag te beantwoorden. Dit had voornamelijk betrekking op queries en welke informatie nodig was om de juiste query op te kunnen stellen. Het volgende fragment is hier een voorbeeld van:

"Dat is een complexe voorwaarde: het gaat om meisjes en ze moeten bij haar in de klas zitten óf ze moeten in de buurt wonen." (IA, Hoofdstuk 10, paragraaf 5.4)

Collecting data

Deze code is maar mondjesmaat gevonden. Wat er is gevonden is ook niet helemaal gelijk aan het verzamelen van data zoals het ISTE/CSTA dat beschrijft in zijn Teacher Resources. Het verzamelen van data was in de teksten voornamelijk een kwestie van informatie opzoeken en vervolgens

opschrijven en geen kwestie van het afnemen van enquêtes. Desondanks horen ook deze fragmenten bij Data Collection.

> *"Maak een lijst van alle public methoden in deze klasse."* (PvS, Opdracht 1, pagina 5)

Data representation

Arranging data for analysis

Van deze code is slechts een enkel fragment gevonden, namelijk de volgende:

> *"Lange zinnen zoals deze zijn niet geschikt voor de analyse, omdat ze te veel informatie in één keer bevatten. Er staat bijvoorbeeld een herhaling van hetzelfde soort informatie, namelijk informatie over drie programma's. Die informatie moet je in drie losse zinnetjes zetten, waardoor de structuur veel simpeler wordt."* (IA, Hoofdstuk 12, paragraaf 5.1)

Organizing/representing data

Fragmenten over dit onderwerp bestonden vooral uit voorbeelden van het representeren van data, zoals het relationele model:

> *"Een goede manier om een database te organiseren is het relationele model."* (IA, Hoofdstuk 10, paragraaf 1.2)

Paralellization

Merging/combining activities

Deze code is eigenlijk niet teruggevonden, al zijn er wel onderdelen van Paralellization teruggevonden. Deze zijn onder deze code gezet omdat er geen andere deelcode van Paralellization was in het huidige codeerschema. De gevonden fragmenten gingen over het plannen van taken en hoe dat zo effectief mogelijk kon:

> *"Sommige taken moet voor andere, sommige taken kunnen parallel uitgevoerd worden, voor andere taken geldt dat ze door meer mensen in te zetten sneller klaar zijn maar er zijn ook taken die zo niet kunt versnellen."* (IA, Hoofdstuk 13, paragraaf 6.1)

CONCLUSIE & DISCUSSIE

Het doel van dit onderzoek was om te ontdekken op welke wijze CT terugkomt in lesmateriaal voor informatica. Hieronder zullen eerst de deelvragen worden besproken die zijn gebruikt om tot een antwoord op de hoofdvraag te komen. Na de deelvragen wordt de hoofdvraag beantwoordt, waarbij telkens wordt ingegaan op de verklaring van de onderzoeksvragen.

De eerste deelvraag van dit onderzoek is: *'Op welke wijze kan bepaald worden welke aspecten van CT in bronnen voorkomen?'*. Doordat empirisch onderzoek naar CT nog maar weinig is uitgevoerd bestond er, voor zover bekend, nog geen methode om aspecten van CT te vinden in (textuele) bronnen. Hierdoor ontstond de noodzaak om een nieuwe methode te ontwikkelen, wat gedaan is in de vorm van een codeerschema waarin alle aspecten van CT, zoals ze beschreven worden in ISTE/CSTA (2011), zijn ingebouwd. Het antwoord op deze deelvraag is dus dat er nog geen methoden hiervoor bestaan, maar dat er in dit onderzoek een instrument is ontwikkeld, in de vorm van een codeerschema, waarmee in textuele bronnen CT aspecten kunnen worden gevonden. Het definitieve antwoord op deze deelvraag is dus dat aspecten van CT gevonden kunnen worden in textuele bronnen door middel van het gebruik van het codeerschema dat in dit onderzoek ontwikkeld is.

Het codeerschema is gemaakt en toegepast door één auteur. Er is daarnaast geen cross-validation toegepast om het codeerschema te valideren. De mogelijkheid bestaat dus dat de resultaten van dit onderzoek subjectief zijn, omdat er niet is gekeken of dezelfde resultaten ook behaald zouden worden wanneer iemand anders het codeerschema ontwierp of toepaste. Daarnaast is het codeerschema gebaseerd op één bron, namelijk de CTLE's van de CSTA/ISTE (2011). Dit kan als gevolg hebben dat, ondanks de validatie met behulp van de volledigheidscheck, het codeerschema onvolkomenheden bevat. Een reden hiervoor kan zijn dat niet alle aspecten van CT evenredig vaak voorkwamen in de CTLE's van de CSTA/ISTE, zoals *Paralellization* die maar een keer voorkwam in de CTLE's. Hierdoor zijn er niet veel codes gevonden bij deze aspecten, waardoor er wellicht onderdelen van deze aspecten niet vertegenwoordigd zijn in het codeerschema. Een andere reden kan zijn dat in de documenten die gebruikt zijn voor de validatie, onderdelen van aspecten niet voorkwamen die ook niet voorkwamen in het codeerschema. Dit heeft als gevolg dat deze niet gevonden werden tijdens de validatie en zodoende niet toegevoegd zijn aan het codeerschema. Het gevolg van deze onvolkomenheden in het codeerschema is dat deze onderdelen van de aspecten van CT ook niet gecodeerd werden in de tekst. Dit betekent dat ze ook missen bij de resultaten van dit onderzoek, waardoor het zo kan zijn dat er nog meer instanties van CT zijn in de onderzochte documenten.

De tweede deelvraag van dit onderzoek luidt: *'Welke inhoudelijke aspecten van CT komen voor in het lesmateriaal?'*. Uit de resultaten van dit onderzoek blijkt dat er een groot verschil is tussen het aantal keer dat bepaalde deelcodes voorkomen. Zo komen de aspecten *Problem decomposition, Simulation* en *Algorithms & procedures* bovengemiddeld vaak voor, terwijl andere aspecten zoals *Paralellization* en *Data collection* ondergemiddeld vaak voorkomen. Daarnaast blijkt ook dat binnen de aspecten van CT de onderdelen van die aspecten onevenredig voorkomen, zoals te zien is in tabel 7 waarin de instanties van de deelcodes van *Algorithms & procedures* beschreven staan. Als laatste is te zien dat het lesmateriaal van IA relatief minder instanties van CT bevat dan het lesmateriaal van PvS. Bij de tweede is de codedichtheid namelijk bijna twee maal zo hoog, wat betekent dat er bijna twee keer zo veel instanties van CT voorkomen per aantal woorden. Het antwoord op deze deelvraag is dat alle aspecten van CT voorkomen in het lesmateriaal, maar dat enkele aspecten bovengemiddeld vaak voorkomen terwijl andere ondergemiddeld voorkomen. Ook is er een groot verschil tussen hoe vaak CT voorkomt in de verschillende lesmaterialen, bij PvS is dat veel vaker dan bij IA.

Een verklaring voor de onevenredigheid van de frequentie van de aspecten van CT kan zijn dat het onderzochte lesmateriaal is toegespitst op deze onderwerpen. Hierdoor komen de aspecten waarover het onderzochte lesmateriaal gaat natuurlijk vaker voor dan de aspecten waar het lesmateriaal niet over gaat. Zo gaat bijvoorbeeld het lesmateriaal van PvS over programmeren en komen aspecten die daar mee te maken hebben, zoals *Algorithms & procedures* en *Problem decomposition*, daar vaak voor. Hetzelfde geldt voor de onderdelen van de aspecten. Daarnaast zijn niet alle onderdelen van IA en PvS onderzocht. Als dit wel wordt gedaan, kan het zo zijn dat de aspecten van CT die nu ondergemiddeld gevonden zijn, daar vaker gevonden worden. Dit zou voor een evenwichtigere verdeling van de aspecten kunnen zorgen, maar daarover is in het licht van dit onderzoek niets te concluderen. Een andere verklaring voor deze onevenredigheid kunnen eventuele onvolkomenheden in het codeerschema zijn. Als hierdoor onderdelen van aspecten niet in het codeerschema zijn opgenomen, zullen deze onderdelen ook niet gevonden zijn in het lesmateriaal, zoals dat is besproken bij de eerste deelvraag.

Een verklaring voor het relatief weinige voorkomen van CT in het lesmateriaal van IA kan zijn dat IA is geschreven op basis van de exameneisen van informatica op het VO (SLO, Stichting Leerplanontwikkeling, 2007). In deze exameneisen wordt CT niet genoemd als eis (Schmidt, 2005), waardoor bij het schrijven van het lesmateriaal CT geen doel was in de stof. Het materiaal van PvS is geschreven met als doel aan te sluiten op modelleervaardigheden van de leerlingen. Modelleren is ook een onderdeel van CT, waardoor het aannemelijk is dat dit een van de redenen is dat CT vaker voorkomt in PvS dan in IA. Dit is bijvoorbeeld terug te zien in het hoge aantal instanties van *Simulation*.

De derde deelvraag van dit onderzoek is: *'Welke variatie is te herkennen tussen de verschillende lesmaterialen ten opzichte van educatieve niveaus?'*. De resultaten van dit onderzoek wijzen uit dat de fragmenten van CT die gevonden zijn in de tekst op verschillende niveaus zitten. Het voornaamste verschil is dat de fragmenten uit IA op lagere niveaus zitten dan die van PvS. De fragmenten uit IA zitten namelijk op de eerste drie niveaus, terwijl de fragmenten uit PvS op alle zes de niveaus zitten. Het antwoord op deze deelvraag is dus dat het lesmateriaal op alle niveaus van de taxonomie van Bloom voorkomen, maar dat deze niveaus per lesmateriaal sterk verschillen.

Een verklaring voor deze verschillen van de niveaus in het lesmateriaal kan zijn dat het lesmateriaal van IA geschreven is aan de hand van de exameneisen van het VO. Deze exameneisen spelen zich af op de eerste drie niveaus van de taxonomie van Bloom. Voorbeelden van het eerste en derde niveau, respectievelijk *Onthouden* en *Toepassen* zijn:

> *'De kandidaat kan de kenmerken van en verschillen tussen real-time systeem, kennissysteem, simulatiesysteem en embedded systeem benoemen.'* (Schmidt, 2005, p. 44)

> *'De kandidaat kan gangbare digitale coderingen van gegevens beschrijven en toepassen.'* (Schmidt, 2005, p. 44)

In de doelstellingen van het lesmateriaal voor PvS komen de hogere niveaus van de taxonomie van Bloom wel aan bod, zoals te zien is in het volgende voorbeeld waar het zesde niveau, Creëren, aan bod komt:

> *'een object-georiënteerd ontwerp maken dat een bruikbare basis vormt voor een implementatie'* (Smetsers, 2012)

De verklaring van het verschil tussen de niveaus van de lesmaterialen op de taxonomie van Bloom zou dus gevonden kunnen worden in het verschil tussen de leerdoelen/exameneisen aan de hand waarvan het lesmateriaal geschreven is.

Met behulp van deze antwoorden op de deelvragen kan er een antwoord gegeven worden op de hoofdvraag van dit onderzoek. Deze hoofdvraag luidt: *'Op welke wijze komt Computational Thinking voor in lesmateriaal op het gebied van informatica?'*. Uit de resultaten van dit onderzoek is op te maken dat CT op het moment al voorkomt in het lesmateriaal op het gebied van informatica. Wel is er een duidelijk verschil te zien tussen de frequentie waarop CT voorkomt in de verschillende lesmaterialen en komen de verschillende aspecten van CT duidelijk niet evenredig vaak voor, waarbij sommige aspecten bovengemiddeld voorkomen en andere ondergemiddeld. Als laatste is er ook te zien dat de niveaus van de CT fragmenten op de taxonomie van Bloom erg verschillend zijn. Zo scoort het lesmateriaal van IA duidelijk lager op deze schaal dan het lesmateriaal van PvS. Het antwoord op deze onderzoeksvraag is dat CT in het huidige lesmateriaal voorkomt, maar dat de aspecten niet evenredig voorkomen en dat de hoeveelheid CT nog laag is in het materiaal. Dit antwoord komt overeen met de verwachtingen van het onderzoek. Daarnaast scoort een deel van het materiaal laag op de taxonomie van Bloom, wat te wijten valt aan de leerdoelen die gesteld worden aan het materiaal.

Aangezien in dit onderzoek gebruik is gemaakt van een kwalitatieve analyse, kunnen de resultaten van dit onderzoek niet gegeneraliseerd worden naar al het lesmateriaal dat beschikbaar is voor informatica. De lesmethode IA en het materiaal van PvS zijn namelijk niet representatief voor al het materiaal dat beschikbaar is. Daarnaast is er in de lesmethode IA niet gekeken naar alle hoofdstukken, waardoor de resultaten van dit onderzoek op het gebied van IA ook niet generaliseerbaar zijn voor heel IA. Wel betekent dit dat er een overzicht is van de aspecten van CT die ten minste voorkomen in IA. Wanneer er onderzoek gedaan zou worden naar de overige onderdelen van IA, zou dit namelijk als gevolg hebben dat er hooguit meer aspecten van CT gevonden worden in deze lesmethode. De instanties van CT die in dit onderzoek gevonden zijn, zitten dus ten minste in deze lesmethode. Om uitspraken te kunnen doen over CT in al het lesmateriaal op het gebied van informatica is het nodig om al deze lesmaterialen te bekijken.

Problem Solving

Zoals in de definitie van CT genoemd wordt, is CT een problem solving proces waarbij problemen op een zodanige wijze geherformuleerd worden dat ze effectief opgelost kunnen worden met de inzet van een information-processing agent. In de gevonden fragmenten op het gebied van CT zijn aspecten terug te zien die worden besproken in eerder werk op het gebied van problem solving. Zo zijn in de fragmenten van PVS aspecten te zien van de barrières en oplossingen voor die barrières die Weigend (2006) beschrijft. Hij identificeert een aantal obstakels waardoor beginnende programmeurs hun intuïtie niet kunnen omzetten in programma code. Deze obstakels zijn als volgt:

- Volgens Weigend gaat een intuïtie voor een oplossing gepaard met een vertrouwen op een goed einde van de opdracht. Door deze intuïtie te moeten specificeren en daarbij tegen onzekerheden aan te lopen, gaat dit vertrouwen verloren. Dit verlies zorgt voor een aarzeling om de intuïtie te specificeren.
- Bij beginners missen er mentale connecties tussen intuïtieve modellen en de corresponderende programmacode
- Een intuïtie kan onbruikbaar zijn voor implementatie, doordat de beginner de juiste programmeer concepten niet kent om de intuïtie te programmeren
- Misverstanden over de betekenis van code kunnen obstakels zijn in het ontwikkelen van programma's.

Weigend geeft aan dat door het geven van mogelijkheden om te reflecteren en te discussiëren over intuïtieve modellen van programmeer concepten de studenten een beter begrip krijgen van hoe ze verbale en non verbale representaties van intuïtieve modellen kunnen gebruiken. Hierdoor kunnen ze een dieper begrip krijgen voor informatica concepten, minder problemen hebben met het communiceren en samenwerken met anderen tijdens software development projecten en kunnen ze slagen in het creëren van computer programma's op basis van hun intuïtie.

In de volgende fragmenten van PvS komen deze barrières van Weigend naar voren. Hierbij wordt de student gevraagd om code heel precies uit te leggen. Dit reflecteren op code wordt door Weigend aangedragen als oplossing tegen de barrières die studenten tegenkomen bij het programmeren. Succesvolle problem solvers zijn in staat om stuken code heel precies uit te leggen ('close tracking'), oftewel ze kunnen goede antwoorden geven op de vraag "verklaar je code". Door studenten dit te laten oefenen kunnen ze beter worden op dit gebied. Voorbeelden van dergelijke fragmenten zijn als volgt:

> *"Leg een boel eieren neer in de wereld en roep Frydes methode act() een aantal keren aan. Experimenteer met de eieren op verschillende plekken. Wat doet de methode act() precies?"* (Opdracht 1)

> *"Beschrijf zo nauwkeurig mogelijk wat de methode act() volgens jou doet."* (Opdracht 1)

De eerste code die Weigend beschrijft, waarbij door het specificeren van een intuïtie het gevoel dat het doel gehaald gaat worden verloren wordt, is terug te zien in het volgende fragment:

> *"Denk in kleine stapjes."* (Opdracht 2)

Bij dit fragment wordt de student gevraagd om zijn intuïtie voor de oplossing van een programmeerprobleem te specificeren en in kleine stapjes op te schrijven. Dit staat gelijk aan de eerste barrière die Weigend beschrijft. Deze opdracht kan dus als gevolg hebben dat de student niet zeker meer weet of het doel gehaald gaat worden. Weigend stelt voor om ruimte voor reflectie in te bouwen bij dit soort onderdelen in het lesmateriaal. In PvS zien we dit helaas niet zo terug. De student krijgt wel een paar tips, maar reflectie op de specificatie wordt pas gegeven aan het eind van deze opdracht, wanneer de student de oplossing al heeft geïmplementeerd. Door meer ruimte voor reflectie in te bouwen bij dergelijke fragmenten het lesmateriaal zouden eventuele barrières die de student ervaart weggenomen kunnen worden.

Naast aspecten die Weigend (2006) beschrijft, zijn er ook aspecten van te zien die Romeike (2008) in zijn model beschrijft. Volgens Romeike is problem solving een groot onderdeel van informatica, maar loopt de interesse en het succes van studenten op dit gebied al jaren terug. Romeike (2008) stelteen aantal problemen met informatica onderwijs op het gebied van problem solving aan de kaak. Volgens Romeike zijn de studenten niet gemotiveerd en geïnteresseerd doordat de problemen die de studenten moeten oplossen:

- Niet relevant zijn voor de studenten: het probleem bestaat niet in de belevingswereld van de student.
- Eigenlijk wiskundige problemen zijn: studenten worden gevraagd om problemen op te lossen op het gebied van de wiskunde, in plaats van problemen op informatica gebied.
- Eigenlijk taken zijn: de student wordt gevraagd om iets te doen waarbij het antwoord triviaal is, waardoor het eigenlijk geen probleem meer is maar meer een taak waarbij aan de student gevraagd wordt om deze uit te voeren.

Romeike geeft aan dat om studenten te motiveren, het belangrijk is om studenten hun eigen probleem te laten vinden binnen een probleemgebied. Ze moeten dan binnen een bepaald kader hun eigen probleem definiëren waarvoor ze een oplossing willen vinden. Daarnaast dient de docent ze aan te sporen om nieuwe problemen te vinden en de studenten te inspireren door prikkelende suggesties te doen. Dit moet leiden tot een intrinsieke motivatie bij de studenten, wat als gevolg heeft dat ze gemotiveerder zijn om het probleem op te lossen en daarin succesvoller zijn. Romeike stelt het volgende model voor dat geïntegreerd moet worden in het informatica onderwijs om te zorgen. Dit model moet zorgen voor een succesvollere manier van het doceren van problem finding in de informatica. Het model doorloopt een drietal fases, welke hieronder worden beschreven. Deze fases dienen vaker per jaar terug te komen in het lesmateriaal:

- Challenge phase: De studenten zoeken binnen een bepaald kader hun eigen probleem dat ze willen oplossen en definiëren dat probleem.
- Problem management phase: De studenten zoeken een oplossing voor het probleem dat ze gekozen hebben.
- Implementation phase: De studenten implementeren hun oplossing. Wanneer ze tegen nieuwe problemen oplopen gaan ze terug naar de problem management phase, om vanuit daar te komen met een oplossing voor het nieuwe probleem.

Doordat elke student zijn eigen, individuele en relevante probleem oplost zijn de studenten intrinsiek gemotiveerd en worden de studenten succesvoller.

In PvS is het mogelijk het globale proces dat Romeike (2008) beschrijft onderscheiden. In opdracht negen wordt een probleemgebied beschreven waarin de student zijn eigen probleem dient te definiëren. Dit probleem dient te voldoen aan een aantal eisen die in de probleembeschrijving worden beschreven. Vervolgens wordt aan de student gevraagd om de oplossing voor dit probleem te implementeren. Hierin zijn de Challenge phase en de Implementation phase uit Romeike's model terug te zien. De Problem management phase wordt hierbij helaas impliciet wel aangehaald, maar er wordt niet expliciet bij stilgestaan. Wanneer deze fase expliciet zou worden aangehaald in de opdrachtomschrijving zouden de studenten wellicht succesvoller kunnen zijn in het uitvoeren van de opdracht.

Uit deze bevindingen kan geconcludeerd worden dat er bij aspecten van CT aandacht besteedt kan worden aan problemen die studenten hebben met problem solving, waardoor de studenten daar minder problemen mee hebben. Dit kan als gevolg hebben dat de studenten CT beter aanleren en kunnen toepassen, omdat ze minder moeite hebben met het problem solving proces dat inherent is aan CT.

VERVOLGONDERZOEK

Door dit onderzoek is er op een aantal gebieden vervolgonderzoek te doen. Ten eerste is het codeerschema dat in dit onderzoek ontwikkeld is een onderwerp dat verder onderzocht kan worden en waarmee vervolgonderzoek uitgevoerd kan worden. Met behulp van dit codeerschema kan er onderzoek gedaan worden naar CT in textuele bronnen, zoals dat gedaan is in dit onderzoek. Een gebied dat bijvoorbeeld interessant is om hiermee te onderzoeken is CT in het lesmateriaal voor het vak informatica op het VO. Dat onderzoek kan bijdragen aan het ontwikkelen van CT lesmateriaal voor het VO, dat vervolgens gebruikt kan worden om de hervormingen die het KNAW (2011) voorstelt door te voeren.

Daarnaast kan er onderzoek gedaan worden naar de volledigheid en de validiteit van het codeerschema. Zoals eerder besproken kan het zo zijn dat het codeerschema onvolkomenheden bevat.

Dit kan opgelost worden door het codeerschema verder te valideren en zo nodig aan te passen. Daarnaast zou het codeerschema kunnen worden omgezet tot een algemener onderzoeksinstrument, waardoor het ook gebruikt kan worden voor onderzoeken die niet te maken hebben met textuele bronnen. Ook is het codeerschema op het moment enkel bruikbaar als descriptief instrument, het kan enkel gebruikt worden om CT te herkennen in reeds geschreven documenten. Wanneer dit schema verder gevalideerd wordt zodat het zeker is dat alle onderdelen van CT erin besproken worden, kan het omgezet worden naar een normatief instrument, dat gebruikt kan worden als checklist voor het schrijven van materiaal voor CT.

Verder kunnen de resultaten van dit onderzoek gebruikt worden als startpunt voor groot onderzoek naar de integratie van CT in het informatica onderwijs en lesmateriaal. De bevindingen kunnen gezien worden als een bevestiging dat CT al voorkomt in het huidige informatica lesmateriaal en kunnen zodoende dienst doen als nulmeting van CT in het lesmateriaal. Door deze nulmeting later te vergelijken met een vergelijkbaar onderzoek naar CT in lesmateriaal dat tegen die tijd ontwikkeld is, kan er gekeken worden wat de vooruitgang is in het lesmateriaal op het gebied van CT.

LITERATUUR

Barr, V., & Stephenson, C. (2011, Maart). Bringing Computational Thinking to K-12: What is Involved and What is the Role of the Computer Science Edication Community? *ACM Inroads, 2*(1), 48-54. doi:10.1145/1929887.1929905

Bryman, A. (2008). *Social Research Methods* (3rd ed.). Oxford: Oxford University Press.

Forehand, & M. (2010). Bloom's taxonomy. *Emerging perspectives on learning, teaching, and technology*.

Forehand, M. (2010). Bloom's taxonomy. *Emerging perspectives on learning, teaching, and technology*.

ISTE; CSTA. (2011). *Computational Thinking Teacher Resources, Second edition.* Opgeroepen op Maart 14, 2013, van CSTA - Computational Thinking: http://csta.acm.org/Curriculum/sub/CompThinking.html

Koninklijke Nederlandse Akademie van Wetenschappen (KNAW). (2012). *Digitale geletterdheid in het voortgezet onderwijs: Vaardigheden en attitudes voor de 21ste eeuw.* Amsterdam: KNAW.

Laarhoven, L., & Vreugdenhil-de Klerk, H. (2012, Juli). *Programmeren (NWI-MOL088) - Bachelor Science, Faculteit der Natuurwetenschappen, Wiskunde en Informatica - 2012 - Radboud Universiteit Nijmegen.* Opgehaald van 2012 - Radboud Universiteit Nijmegen: http://www.studiegids.science.ru.nl/2012/science/prospectus/natural_science_bachelor/courses/course/28381/

Mayring, P. (2000, Juni). Qualitative content analysis. *Forum qualitative sozialforschung/Forum: qualitative social research, 1*(2).

Romeike, R. (2008). What's my challenge? The forgotten part of problem solving in computer science education. *Informatics Education-Supporting Computational Thinking*, 122-133.

Schmidt, V. (2005). *Handreiking schoolexamen Informatica.* Enschede: Stichting Leerplanontwikkeling.

SLO, Stichting Leerplanontwikkeling. (2007). *Vakdossier informatica 2007.* Enschede: SLO.

Weigend, M. (2006). From intuition to programme. *Informatics Education–The Bridge between Using and Understanding Computers.*, 117-126.

Wing, J. M. (2006, Maart). Computational Thinking. *Communications of the ACM, 49*(3), 33-35.

Wing, J. M. (2011). *Research Notebook: Computational Thinking: What and Why.* Opgeroepen op april 2013, 15, van SCHOOL OF COMPUTER SCIENCE, Carnegie Mellon: http://link.cs.cmu.edu/article.php?a=600