Universiteit Utrecht

Computationeel denken in het basisonderwijs

Een analyse van de methoden ´Programmeren in het PO´, ´Bomberbot´ en ´Codekinderen´ op eigenschappen van computationeel denken in het basisonderwijs.

Simone de Jager

5891353

Bachelorscriptie Kunstmatige Intelligentie, UU

7.5 ECTS

Rianne van Lambalgen (1e beoordelaar)

Jan Broersen (2e beoordelaar)

2 juli 2021

s.j.dejager@students.uu.nl

Samenvatting

Om in onze huidige kennismaatschappij kritisch naar technische systemen te kunnen kijken, moeten wij meer weten over hoe ze werken. Door computationeel denken in het onderwijs aan te leren, wordt een fundering voor kunstmatige intelligentie gelegd. In deze scriptie wordt onderzocht hoe een aantal bestaande methoden voor het basisonderwijs computationeel denken bij de leerlingen aanleert en worden ze met elkaar vergeleken. De onderzoeksvraag die voor dit literatuuronderzoek is opgesteld, luidt : “Welke eigenschappen van computationeel denken zijn belangrijk in het onderwijs en hoe worden deze door de huidige methoden geïmplementeerd wanneer gekeken wordt naar de beschreven methode?”.

Uit een vergelijking van verschillende definities van computationeel denken blijkt dat de meta-analyse van Selby en Woollard (2013) vijf belangrijke eigenschappen noemt, namelijk: abstract denken, in stukjes denken, algoritmisch denken, evalueren en generaliseren.

Uit een kwalitatieve analyse van de methoden ‘Programmeren in het PO’, ‘Bomberbot’ en ‘Codekinderen’, waarin gekeken werd naar deze vijf eigenschappen, blijkt dat al deze methoden werken aan computationeel denken in het basisonderwijs. Er valt echter niet te zeggen welke methode dit meer of minder doet, aangezien de eigenschappen niet tegen elkaar op te wegen zijn en de methoden allemaal verschillende cognitieve leertheorieën toepassen als manieren om de leerstof aan de leerlingen aan te bieden. Het is dus afhankelijk van de klas en leerkracht welke methode het beste computationeel denken aan die leerlingen leert.

Inhoudsopgave

1. Inleiding 4

2. Eigenschappen computationeel denken 7

2.1. Verschillende definities 7

2.2. Belangrijkste eigenschappen en betekenis 8

3. Hoe leren kinderen 10

4. Methoden in het onderwijs 12

4.1. Beschrijving 'Programmeren in het PO' 12

4.2. Beschrijving 'Bomberbot' 13

4.3. Beschrijving 'Codekinderen' 14

5. Resultaten 15

5.1. Tabel analyse eigenschappen computationeel denken 15

5.2. Toelichting tabel 16

5.2.1. Abstract denken 16

5.2.2 In stukjes denken 17

5.2.3. Algoritmisch denken 18

5.2.4. Evalueren 19

5.2.5. Generaliseren 19

5.3 Methoden en leertheorieën 20

6. Discussie 22

7. Conclusie 24

8. Literatuur 26

1. Inleiding

Onze maatschappij is tegenwoordig een kennismaatschappij, waarin iedereen een bijdrage levert aan de digitale informatie en communicatie (KNAW, 2012). Het is daarom van belang dat iedereen leert omgaan met deze digitale middelen. Volgens de KNAW (Koninklijke Nederlandse Akademie van Wetenschappen) gebeurde dit voor 2012 echter nog niet genoeg in het onderwijs; het vak informatica op de middelbare school bewoog bijvoorbeeld niet genoeg mee met nieuwe innovatietechnieken. Daarom werd een commissie samengesteld genaamd 'Informatica in het voortgezet onderwijs'. Deze commissie had het doel het gehele onderwijs met betrekking tot digitale informatie en communicatie te herzien (KNAW, 2012). Volgens het Ministerie van Onderwijs, Cultuur en Wetenschap moeten scholen sinds 2014 inderdaad meer aandacht besteden aan verschillende 21^{e} eeuwse vaardigheden. Dit noemden zij 'digitale geletterdheid' (SLO, 2020b). Onder digitale geletterdheid vallen de begrippen 'computational thinking', 'mediawijsheid', 'ICT-basisvaardigheden' en 'informatievaardigheden' (Van Ramele, 2019).

Een belangrijk begrip binnen digitale geletterdheid is computationeel denken, vaak in de literatuur 'computational thinking' genoemd (Thijs et al., 2014). Deze term werd voor het eerst gebruikt door Wing (2006). Zij gaf aan dat computationeel denken betrekking heeft op technische en wiskundige problemen, maar ook op situaties uit het dagelijks leven waarvoor complexe software systemen nodig zijn. Zo laat het ook leerlingen uitdagingen zo formuleren dat ze op te lossen zijn met computers (Kennisnet, z.d.). Dit valt te leren door, maar is ook toepasbaar in programmeren (Voogt, Brand-Gruwel & van Strien, 2017). Echter, computationeel denken is ook toepasbaar in gebieden buiten programmeren. Het zorgt er namelijk ook voor dat leerlingen met meer vertrouwen allerlei complexe problemen kunnen aangaan, doordat ze het probleem nu kunnen opdelen in subproblemen en hier een efficiënte oplossingsstrategie voor leren te vinden (Kennisnet, 2019).

In de meta-analyse van Selby en Woollard (2013) worden verschillende definities van computationeel denken met elkaar vergeleken en wordt een nieuwe definitie voorgelegd die in lijn ligt met probleemoplossend denken. Zij stellen dat het om problemen op te kunnen lossen met computationeel denken nodig is om abstract te kunnen denken, om in stukjes te kunnen denken, om algoritmisch te denken, om te kunnen evalueren en om te kunnen generaliseren (Selby & Woollard, 2013). In het volgende hoofdstuk van deze scriptie zal worden uitgelegd wat deze concepten precies inhouden.

Het doel van deze scriptie is om te verifiëren of, en zo ja hoe, methodes die claimen kinderen computationeel denken aan te leren daadwerkelijk aandacht besteden aan alle eigenschappen ervan. Op basis hiervan kunnen de methoden met elkaar worden vergeleken en kan worden bepaald welke de meeste eigenschappen van computationeel denken behandelt en wat verschillende manieren zijn om dit tot stand te brengen. Dit is belangrijk, omdat al deze kernpunten samen de 'grammatica' van het programmeren vormen (Kennisnet, 2016). Dus, pas als je kennis hebt over (al) deze begrippen kun je met programmeren aan de slag, anders niet. De resultaten van deze scriptie zullen inzicht bieden in welke programma's hierin slagen en hoe dit verbeterd kan worden bij andere methoden. Om in te kunnen gaan op de manier waarop de methoden de leerlingen computationeel denken aanleren, zal in deze scriptie ook worden gekeken naar hoe kinderen leren (dus naar cognitieve leertheorieën) en in hoeverre de methoden daarop inspelen.

De resultaten van deze scriptie zijn relevant voor de toekomst van kunstmatige intelligentie. Het is belangrijk om te leren hoe systemen werken (Bosse, 2019). Alleen dan kunnen we namelijk begrijpen hoe we om moeten gaan met de resultaten van een programma, iets wat tegenwoordig erg relevant is doordat er steeds meer programma's bestaan, maar nog veelal beperkt blijft. Het feit dat wij vaak niet begrijpen hoe systemen werken, zorgt ervoor dat ze regelmatig niet worden toegepast. Dit belemmert de vooruitgang in techniek. Ook om kritisch naar deze systemen te kunnen kijken, moeten wij meer weten over hoe ze werken. Door te investeren in kunstmatige intelligentie, zal onze kennis over deze systemen groeien (Bosse, 2019).

Wing (2006) zegt dat we kunstmatige intelligentie nodig hebben om mensen van computers te onderscheiden. Gadanidis (2016) heeft geschreven over de overeenkomsten tussen kunstmatige intelligentie, computationeel denken en wiskundeonderwijs. Hij ondervond dat de factoren agentschap, modellen en abstractie in al deze gebieden van toepassing zijn (Gadanidis, 2016). Als leerlingen leren computationeel te denken, zullen ze dus ook worden voorbereid op deze onderdelen van kunstmatige intelligentie.

Door op jonge leeftijd te leren schematiseren, worden leerlingen beter in rekenen en wiskunde (Poland, van Oers & Terwel, 2010). Er wordt aangeraden om te werken aan een voortdurende ondersteuning van computationeel denken, al vanaf jonge leeftijd (Lee et al., 2011). Daarom is door het Ministerie van Onderwijs de keuze gemaakt om al op de basisschool computationeel denken aan te leren en focust deze scriptie op die leeftijdscategorie (SLO, 2020b).

Op basis van alle voorgaande informatie is een onderzoeksvraag opgesteld. Deze luidt:

Welke eigenschappen van computationeel denken zijn belangrijk in het onderwijs en hoe worden deze door de huidige methoden geïmplementeerd wanneer gekeken wordt naar de beschreven methode?

In deze studie wordt bij drie methoden gekeken of zij al deze aspecten toepassen: ‘Programmeren in het PO (Primair Onderwijs)’, ‘Bomberbot’ en ‘Codekinderen’. Hoewel deze drie methodes anders werken, claimen ze alle drie computationeel denken aan te leren aan leerlingen in het basisonderwijs en geven ze alle drie inzicht in de precieze inhoud van de lessen. Dit zorgt dan ook voor een kwalitatieve analyse.

De verwachting voor de resultaten is dat er waarschijnlijk meerdere definities van computationeel bestaan, waardoor het lastig te bepalen is welke eigenschappen belangrijk zijn. Verder wordt verwacht dat niet alle methoden in dezelfde mate aandacht besteden aan de verschillende aspecten van computationeel denken, onder andere doordat de definitie niet eenduidig is. Echter, het kan zo zijn dat de manier waarop de leerstof wordt aangeboden, bijvoorbeeld bij een methode die erg uitgebreid de verschillende aspecten van computationeel denken ondersteunt, minder aansluit bij de cognitieve leertheorieën dan hoe een andere methode dit aanpakt. Er zijn dus best veel factoren die invloed hebben op de potentiële effectiviteit (op het gebied van computationeel denken) van de programma’s, en het antwoord op de onderzoeksvraag zal dan ook genuanceerd zijn.

In het vervolg van deze scriptie wordt allereerst nagegaan welke eigenschappen van computationeel denken moeten worden behandeld in een methode die leerlingen succesvol tot computationeel denken aanzet en waarom. Vervolgens wordt bekeken welke cognitieve leertheorieën bekend zijn en hoe dit op verschillende manieren didactisch kan worden ingezet. Daarna wordt van een aantal methoden die aangeven kinderen met basisschoolleeftijd tot computationeel denken aan te zetten uitgezet hoe zij dit doen, waarna een analyse wordt gepresenteerd waaruit duidelijk zal worden in hoeverre de methoden de verschillende aspecten van computationeel denken toepassen. Deze scriptie wordt afgesloten met een discussie en conclusie.

2. Eigenschappen computationeel denken

2.1. Verschillende definities

Een belangrijk begrip binnen digitale geletterdheid is computationeel denken, vaak in de literatuur ‘computational thinking’ genoemd (Thijs et al., 2014). Deze term werd voor het eerst gebruikt door Wing (2006). Zij gaf aan dat computationeel denken betrekking heeft op technische en wiskundige problemen, maar ook tot situaties uit het dagelijks leven waarvoor complexe software systemen nodig zijn. De manier waarop wordt nagedacht bij computationeel denken overlapt met het soort denken over kunstmatige intelligentie (Zeng, 2013). Om tot kunstmatige intelligentie te kunnen komen, is het namelijk noodzakelijk dat de computationele intelligentie van de machine of het programma wordt begrepen. Inzichten als abstract denken en probleemoplossend denken hebben dan ook direct betrekking op kunstmatige intelligentie (Zeng, 2013).

Volgens de meta-analyse over de definitie van computationeel denken van Selby en Woollard (2013) is er enige onenigheid tussen mensen die vinden dat computationeel denken erg precies of juist breed gedefinieerd zou moeten worden. Een argument om de definitie breed te houden is dat op die manier de focus meer gelegd kan worden op de manier waarop computationeel denken geïmplementeerd zou moeten worden. Aan de andere kant wordt gezegd dat een precieze definitie ervoor zorgt dat beter kan worden nagegaan of het goed in het curriculum wordt geïmplementeerd (Selby & Woollard, 2013). In deze literatuurstudie is juist dit laatste erg belangrijk, omdat in de analyse wordt nagegaan in hoeverre alle aspecten van computationeel denken aan bod komen. Omdat er verschillen in definities bestaan, zullen er nu een aantal met elkaar worden vergeleken, om tot een gangbare definitie te komen waarmee methoden kunnen worden geanalyseerd.

Volgens het SLO (Stichting Leerplan Ontwikkeling) valt computationeel denken onder de doelen voor digitale geletterdheid (SLO, 2020a). Computationeel denken heeft volgens hen betrekking op de denkprocessen die bij onder andere een probleemformulering en -analyse nodig zijn om problemen met behulp van ICT-technieken op te lossen. Kennisnet (2019) geeft aan dat computationeel denken inderdaad te maken heeft met programmeren, maar daarnaast nog veel breder is. Zo laat het ook leerlingen uitdagingen zo formuleren dat ze op te lossen zijn met computers. Om oplossingen te vinden moeten ze informatie logisch kunnen structureren, analyseren, abstraheren en weergeven. Verder bevat computationeel denken ook het tot een antwoord kunnen komen door de meest effectieve en efficiënte stappen te vinden. Dit proces kan worden gegeneraliseerd naar andere toepassingen. Samenvattend ontwikkel je met computationeel denken ruimtelijk inzicht en

probleemoplossend vermogen. Dit is toepasbaar in veel meer gebieden dan alleen programmeren. Het zorgt er namelijk ook voor dat leerlingen met meer vertrouwen allerlei complexe problemen kunnen aangaan (Kennisnet, 2019).

Hu schreef in 2011 over de verschillen in definities van computationeel denken. Zij kwam uiteindelijk op een definitie die vooral te maken had met de manier van denken die komt kijken bij computationeel denken. Zo moet deze manier van denken logisch, algoritmisch, wetenschappelijk, wiskundig, analytisch, technisch en creatief zijn om problemen te kunnen oplossen, systemen te kunnen automatiseren en modellen te kunnen maken (Hu, 2011).

In de meta-analyse van Selby en Woollard (2013) worden verschillende definities met elkaar vergeleken en wordt een nieuwe definitie voorgelegd die in lijn ligt met probleemoplossend denken. Deze definitie gaat als volgt: “Computational thinking is a focused approach to problem solving, incorporating thought processes that utilize abstraction, decomposition, algorithmic design, evaluation, and generalizations”. In andere woorden, om problemen op te lossen met computationeel denken is het nodig om abstract te denken, om in stukjes te kunnen denken, om algoritmisch te denken, om te kunnen evalueren en om te kunnen generaliseren (Selby en Woollard, 2013).

2.2. Belangrijkste eigenschappen en betekenis

Aangezien de definitie die door Selby en Woollard (2013) is geformuleerd, gebaseerd is op 22 artikelen, door honderden publicaties wordt geciteerd en samenvat wat ook in de andere hierboven beschreven definities wordt genoemd, is besloten om in de analyse van deze literatuurstudie gebruik te maken van deze definitie van computationeel denken. Om duidelijk te maken wat deze definitie precies inhoudt, zullen nu alle beschreven eigenschappen van computationeel denken worden uitgelegd. Deze eigenschappen zijn het vermogen om abstract te denken, om in stukjes te kunnen denken, om algoritmisch te denken, om te kunnen evalueren en om te kunnen generaliseren.

Volgens Denning (Ubiquity, 2007) komt abstractie veel voor in programmeren, maar ook op andere gebieden. Wing (2006) gaf aan dat abstractie een belangrijk concept binnen computationeel denken is. Bij abstractie wordt complexiteit vermeden en doordat er verschillende lagen van abstractie denkbaar zijn leer je hier met computationeel denken de relaties tussen te zien (Selby & Woollard, 2013). De methode ‘programmeren in het PO’ beschrijft in de definitie voor abstractie dat dit ook gaat om het veralgemeniseren van problemen, waardoor je een oplossing kunt vinden die voor meerdere problemen bruikbaar is (Kennisnet, 2016).

Het ‘in stukjes kunnen denken’ wordt vaak decompositie genoemd. Dit houdt simpelweg in dat je een probleem in verschillende deelproblemen kunt opdelen (Kennisnet, 2016). Dit heeft betrekking tot meerdere leerdoelen van het SLO. Doordat een probleem in deelproblemen is opgedeeld kunnen delen van algoritmes en procedures gegeneraliseerd worden, waardoor dit (deel)probleem nu ook bij andere problemen toegepast kan worden. Verder kunnen binnen de automatisering nu mogelijke oplossingen worden geïdentificeerd, geanalyseerd en geïmplementeerd met als doel de meest effectieve en efficiënte oplossing te vinden, doordat deze oplossingen voor deelproblemen makkelijker te vinden zijn dan voor een groter probleem (Kennisnet, 2016).

Een algoritme is een reeks instructies die wanneer deze stap voor stap in de juiste volgorde uitgevoerd worden tot een doel leiden (Kennisnet, 2016). Dit is nodig om een computerprogramma te kunnen schrijven in code, maar kan ook een gegeneraliseerde efficiënte oplossing bieden om tot een oplossing te komen.

Evalueren heeft te maken met analytisch kunnen denken. Het is hiervoor nodig om bijvoorbeeld te kijken naar waarom een bepaalde denkstap in het oplossen van een probleem of voor het maken van een programma eigenlijk is gemaakt, te checken of er niet vaak is verondersteld dat iets geldt zonder concreet bewijsmateriaal te vinden, te controleren of alles nog klopt vanuit verschillende kanten van een verhaal etc., om tot een sterk onderbouwd antwoord op een probleem te komen (Hu, 2011). Dit is dus een eigenschap van computationeel denken die toepasbaar is in programmeren, maar ook in het oplossen van een probleem in het dagelijks leven.

Generalisatie houdt in dat patronen worden herkend en kunnen worden hergebruikt in andere toepassingen (Selby & Woollard, 2013). Dit is nodig binnen programmeren, omdat er vaak gebruik wordt gemaakt van herhaling om computers zelf verder te laten werken aan het zoeken van een oplossing (Kennisnet, 2016). Dit helpt dus om programma’s of taken te kunnen automatiseren, maar ook om dit efficiënt te laten gebeuren. In het dagelijks leven kan dit ook helpen om oplossingen voor een bepaald probleem uit te kunnen testen in een andere context. Zo kan bijvoorbeeld een tas helpen meer boodschappen mee naar huis te nemen, maar kan een tas ook helpen om meer kleding mee te kunnen nemen op reis.

3. Hoe leren kinderen

Om deze eigenschappen van computationeel denken toe te kunnen passen in een methode, moet ook worden gekeken naar hoe leerlingen leren. Alleen wanneer de eigenschappen op een aansprekende manier aan de leerlingen worden geïntroduceerd, kunnen ze daadwerkelijk de kennis en vaardigheden op de beoogde manier aanleren. Daarom is het belangrijk kort te bespreken wat de bekendste cognitieve leertheorieën zijn.

Een grote naam binnen de cognitieve leertheorieën is Jean Piaget. Volgens Piaget (1964) leren leerlingen niet vanzelfsprekend, maar construeren ze kennis actief door interactie met hun omgeving. Ze leren dus door te experimenteren. Hij heeft een model ontwikkeld waarin verschillende stadia van cognitieve ontwikkeling voorkomen, genaamd de sensomotorische, preoperationele, concreet operationele en formeel operationele fase. De snelheid waarin deze stadia worden doorlopen verschilt per kind en cultuur. Gemiddeld zitten kinderen van 0-2 jaar in de sensomotorische fase. Dit houdt in dat ze leren door fysieke acties, die vaak reflexief en instinctief zijn na de geboorte en steeds meer richting symbolisch denken gaan. Kinderen van ongeveer 2 tot 7 jaar oud zitten in de preoperationele fase. In deze fase kunnen kinderen leren door taal te gebruiken. Met woorden en beelden kunnen kinderen steeds meer symbolisch denken en gaat leren verder dan dat dit alleen fysiek en sensorisch wordt aangestuurd. In de concreet operationele fase zitten veelal kinderen tussen de 7 en 11 jaar oud. In deze fase van cognitieve ontwikkeling leren leerlingen logisch te redeneren binnen concrete gebeurtenissen en kunnen ze objecten binnen bepaalde contexten plaatsen. Vanaf 11 jaar zitten kinderen in de formeel operationele fase, waarin ze veel abstracter, idealistischer en logischer kunnen denken (Piaget, 1964).

Er is in de loop van de jaren wel kritiek op Piaget’s model gekomen (Kijk op ontwikkeling, 2014). Zo vonden onder andere de sociaal-culturele theoretici dat Piaget te weinig naar de functie van taal, de rol van de cultuur en opvoeders en het belang van de omgeving keek. Zij vinden namelijk, tegen Piaget’s overtuigingen in, dat de cognitieve ontwikkeling sterk door omgeving en interactie wordt beïnvloed. Op basis van deze kritiek zijn een aantal aangepaste modellen ontstaan: de sociaal-culturele theorie die juist ingaat op deze omgevingsfactoren, de Core Knowledge theorie wat de nadruk legt op de invloed van evolutie en genen als basis om vanuit te kunnen leren en de informatieverwerkingstheorie wat de hersenen ziet als de hardware en datgene wat men met deze informatie vervolgens kan vormt de software.

Andere, voornamelijk didactische, theorieën die invloed hebben op de manier waarop leerlingen leren hebben bijvoorbeeld betrekking op feedback geven, competentie, autonomie, relatie, intrinsieke en extrinsieke motivatie en de manier waarop de lessen zijn opgesteld (Bakx, 2015). Aangezien de meeste van deze concepten (net als de hiervoor beschreven theorieën als kritiek op Piaget) vooral door de leerkracht worden beïnvloed, zijn ze niet allemaal even relevant voor deze analyse en zullen ze nu maar heel kort worden uitgelegd.

Het niveau waarop feedback wordt gegeven is belangrijk voor het leerproces, omdat dit mede bepaalt wat de attitude van de leerling tegenover de leerstof zal zijn (Bakx, 2015). Er kan sprake zijn van feedback op taak-, proces-, zelfregulerend- en persoonsniveau. Feedback op taakniveau geeft aan of de leerlingen de taak goed of fout hebben gemaakt. Feedback op procesniveau heeft betrekking op de manier waarop een leerling de taak heeft aangepakt. Feedback op zelfregulerend niveau blijkt erg effectief en geeft leerlingen ‘tools’ om bij toekomstige taken zelf na te kunnen gaan of ze ‘goed bezig zijn’. Feedback op de persoon blijkt het minst effectief als het gaat om leeropbrengsten van leerlingen. Het is feedback wat gaat over de leerling zelf, zoals in het voorbeeld ‘Wat knap van jou!’(Bakx, 2015).

De zelfdeterminatietheorie geeft aan dat het voor een leerling essentieel is dat hij/zij zich competent voelt, autonomie krijgt en een goede relatie heeft met de leerkracht, voordat het kind kan leren (Bakx, 2015). Ook kunnen intrinsieke en extrinsieke motivatie een grote impact hebben op het leren. Bij intrinsieke motivatie wil een kind zelf graag leren. Bij extrinsieke motivatie is er een extrinsieke (dus vanaf buiten het kind) factor die impact heeft op het willen leren van het kind, bijvoorbeeld een leerkracht die zegt dat het moet of een beloning als het kind de opdrachten af heeft.

Volgens Bakx, Ros en Bolhuis (2017) wordt bij opbrengstgericht werken gewerkt vanuit meetgegevens. Bij handelingsgericht werken worden daarentegen de onderwijsbehoeften van leerlingen centraal gesteld. Deze handelingswijzen kunnen goed verenigd worden en voor passend onderwijs is het verstandig om te kijken naar wat de klas nodig heeft (Bakx, Ros en Bolhuis, 2017).

4. Methoden in het onderwijs

Om de verschillende methoden die zullen worden geanalyseerd op het toepassen van computationeel denken uit te kiezen, waren een aantal factoren belangrijk. Zo moest de methode in ieder geval claimen computationeel denken aan leerlingen met basisschoolleeftijd aan te leren, aangezien dit hetgeen is waar in deze analyse naar gekeken wordt. Verder moet de methode zodanig uitgebreid zijn beschreven, dat een analyse op de verschillende factoren van computationeel denken mogelijk is. Zo moet in ieder geval ergens staan beschreven hoe de methode/leerlijn werkt en welke onderdelen aan bod komen. Ook heeft het een voorkeur als er voorbeelden van lessen of oefeningen worden beschreven en als wordt aangegeven wat de bijbehorende didactiek is (de manier waarop iets wordt uitgelegd of hoe lang de leerlingen met een bepaald onderdeel bezig zijn bijvoorbeeld). De verschillende methoden die aan deze factoren voldeden en daarom voor deze analyse zijn uitgekozen, zijn: de leerlijn ‘programmeren in het PO’, ‘Bomberbot’ en ‘Codekinderen’.Voordat kan worden geanalyseerd hoeveel deze methoden computationeel denken toepassen zullen nu eerst de verschillende methoden worden beschreven.

4.1. Beschrijving ‘Programmeren in het PO’

Vanuit het SLO wordt gelinkt naar de leerlijn ‘programmeren in het PO’ van Kennisnet (2016). Deze leerlijn is inzetbaar in alle groepen in het basisonderwijs en geeft doelen en activiteiten om de leerlingen te leren programmeren. Er wordt genoemd dat de leerlijn ook gericht is op het bevorderen van computationeel denken. Een school mag zelf keuzes maken in bijvoorbeeld het uitbreiden of de manier van aanbieden van de verschillende elementen in de leerlijn. Dit kunnen ze bijvoorbeeld laten afhangen van de aanwezige voorkennis van de leerlingen.

De leerlijn is opgebouwd aan de hand van tien begrippen die samen leerlingen laten programmeren. Per begrip staan voor elke bouw vervolgens één of meerdere lessen uitgewerkt die de leerkracht stap voor stap kan volgen. De begrippen die in de leerlijn aan bod komen zijn: algoritmes, decompositie, patronen, herhaling, fouten, voorwaarden, abstractie, functie, variabele en representatie. Deze begrippen nemen in complexiteit toe, doordat ze steeds abstracter worden en meer met de overige begrippen te maken hebben. Per begrip wordt ook een uitleg gegeven waarna de verschillende leerdoelen van het SLO aan het begrip zijn gekoppeld. Niet elk begrip heeft een (aantal) les(sen) voor alle drie de bouwen; soms kan het voorkomen dat bijvoorbeeld alleen lessen voor de onder- of middenbouw beschikbaar zijn. Volgens Kennisnet is het namelijk niet altijd nodig om op elk begrip extra

in te gaan, omdat begrippen als decompositie in meerdere andere lessen terugkomen, of soms hoeft hier niet extra op in worden gegaan omdat een begrip al uitgebreid was besproken.

4.2. Beschrijving 'Bomberbot'

Bomberbot is een organisatie die codekampen en een lesmethode aanbiedt voor alle leerjaren van de basisschool tot en met de eerste twee klassen van het VO (Voortgezet Onderwijs) (www.bomberbot.com). De beschrijving en analyse van deze scriptie hebben betrekking op de lessen die Bomberbot aanbiedt voor het primair onderwijs. Vanaf de zomer van 2021 gaat Bomberbot over naar een nieuw lesprogramma digitale geletterdheid, waarin per leerjaar op verschillende aspecten van digitale geletterdheid wordt ingegaan. Hier is ook een gedeelte genaamd 'programmeren voor groep 5 t/m 8', wat specifiek ingaat op computationeel denken in deze groepen. (Bomberbot, z.d.). Door leerkrachten van uitgebreide lesplannen te voorzien hoeven zij geen voorkennis in programmeren te hebben om toch met de leerlingen hiermee aan de slag te kunnen (Bomberbot, z.d.).

Om meer te weten te komen over Bomberbot hebben zij een aantal interne documenten opgestuurd. Deze worden onder 'persoonlijke communicatie' vermeld, aangezien weinig over de oorsprong van de documenten te vinden is. Kokoa Finland heeft een pedagogische en evaluatieve analyse uitgevoerd op de methode van Bomberbot (persoonlijke communicatie). Volgens hen scoort Bomberbot 640 punten van de 700 op computationeel denken. De voornamelijkste onderwerpen die bij Bomberbot aan bod komen zijn volgens hen het scripten, algoritmes bouwen en begrijpen van programmeren (persoonlijke communicatie).

Voor alle Bomberbotlessen wordt ongeveer dezelfde opbouw gebruikt. Dit begint met een introductie, vervolgens gaan de leerlingen aan de slag met (unplugged) games en missies (persoonlijke communicatie document genaamd 'Bomberbot pedagogical framework'). Het programma geeft hints en mogelijkheden om een level te testen en opnieuw te proberen. Leerkrachten kunnen de voortgang van de leerlingen online in de gaten houden, doordat ze kunnen zien hoeveel levels de leerlingen al hebben gemaakt en hoeveel sterren ze daarvoor hebben gekregen. Het laatste onderdeel van een les is het ontwerpen en maken van hun eigen level. Deze kunnen ze delen met hun klasgenoten, waardoor ze zowel hun eigen level als die van klasgenoten kunnen evalueren (persoonlijke communicatie).

Voor de groepen 5 tot en met 8 komen in het eerste jaar dat met Bomberbot wordt gewerkt de volgende onderwerpen aan bod: wat is programmeren, sequenties, for loops, relatieve richtingen, algoritmes, functies, variabelen, until loops, conditionals, programma flow en unplugged level maker (Bomberbot, z.d.). In het volgende jaar zijn de onderwerpen:

debugging, functies 2, nested loops, functie hergebruik, for loops in functies, recursieve functies, probleem oplossen, samenwerken.

4.3. Beschrijving 'Codekinderen'

Ook Codekinderen laat leerlingen uit het basisonderwijs en de onderbouw van het VO oefenen met computationeel denken (Codekinderen, z.d.). Zij hebben de verschillende lessen onderverdeeld in drie categorieën: unplugged, maken en programmeren. Onder 'unplugged' vallen de activiteiten waar je geen computer voor nodig hebt. Zo worden de principes van het programmeren heel tastbaar gemaakt. Onder 'maken' beschrijven zij activiteiten die te maken hebben met elektronica, maar die verder fysiek zijn, zoals het maken van een stopmotion. Bij de activiteiten onder 'programmeren' worden opdrachten gegeven om leerlingen echte programmeercode te laten schrijven. Er is geen vaste volgorde te vinden waarin de leerkrachten de verschillende activiteiten kunnen doorlopen (Codekinderen, z.d.).
Achter de verschillende activiteiten van Codekinderen staat aangegeven of dit voor het PO of VO bedoeld is (www.codekinderen.nl). Per activiteit staat ook in de lesbrief beschreven voor welke groepen de activiteit uitermate geschikt is en wat precies de stappen zijn die de leerkracht of klas precies in de les kunnen doorlopen. Ook zijn bij elke activiteit een docentenhandleiding en meestal ook een werkblad bijgevoegd.

Codekinderen heeft geen eigen programmeerplatform. In plaats hiervan beschrijven ze lessen die soms (vooral bij de activiteiten onder 'programmeren') kunnen worden uitgevoerd op andere platforms. Zo wordt er bijvoorbeeld gebruik gemaakt van Scratchjr., Codeuur, Lightbot en Beebot. Een aantal activiteiten die in de de unplugged activiteiten aan bod komen zijn: robot- of codetaal leren, binair je naam schrijven en muffins maken. Bij de maakactiviteiten komt onder andere aan bod: elektrische klei maken, een animatie, stopmotion of pixelart maken, aan de slag gaan met een chatbot en een hologram maken. Onder 'programmeren' valt bijvoorbeeld: een verhaal maken, flappybird maken en logisch denken met Ko de Kraker.

5. Resultaten

5.1. Tabel analyse eigenschappen computationeel denken

Om te onderzoeken of de verschillende methoden alle onderdelen van computationeel denken aan de leerlingen aanleren, is onderstaande tabel opgesteld. Hierin wordt per onderdeel uit de definitie van computationeel denken nagegaan of, en zo ja hoe, de methode dit toepast. De focus ligt hierbij op de lessen voor het PO. In de analyse over Bomberbot wordt gekeken naar de lesmethode programmeren voor de groepen 5-8, dat twee lesjaren beslaat. Voor de analyse over Codekinderen, wordt per aspect van computationeel denken gekeken naar de inhoud van alle verschillende lesactiviteiten die op de website staan, aangezien de lessen staan ingedeeld onder ‘unplugged’, ‘maken’ en ‘programmeren’.

Deze analyse kent geen concrete score aan de verschillende methoden toe. In plaats daarvan wordt kwalitatief naar de inhoud van alle beschikbare lessen gekeken. Wel worden in de Tabel 1 plusjes en minnetjes gegeven bij alle specifieke onderdelen, om overzicht te geven. In 5.2 volgt uitgebreide toelichting hierop. Vaak houdt een plusje in dat dit concept concreet wordt behandeld. +/- houdt veelal in dat het concept wel aan bod komt, maar er niet diep op wordt ingegaan of niet concreet benoemd wordt. Een minnetje betekent vaak dat dit concept niet of niet duidelijk wordt behandeld.

Tabel 1

Overzicht van concepten computationeel denken in de verschillende methoden

	Programmeren in het PO	**Bomberbot**	**Codekinderen**
Abstract denken	+	+/-	-
In stukjes denken	+/-	+/-	-
Algoritmisch denken	+	+	+/-
Evalueren	-	+/-	--
Generaliseren	+/-	++	+/-

5.2. Toelichting tabel

5.2.1. Abstract denken

Programmeren in het PO (Kennisnet, 2016) heeft voor zowel de middenbouw als de bovenbouw een les beschreven die abstract denken stimuleert. Voor de middenbouw geldt dat deze les gebruik maakt van decompositie, patroonherkenning, abstractie en algoritmes, om tot de algemene regel van het optellen van alle getallen tot 200 te komen. De leerlingen leren na te denken over een algemene regel die geldt voor het oplossen van een dergelijk probleem. In een les voor de bovenbouw gaan de leerlingen hun dag abstraheren naar een algemeen schema. Door algemene woorden te gebruiken (‘ontbijt’ in plaats van ‘boterham’ bijvoorbeeld), is dit verhaal voor veel meer leerlingen in te vullen. Door een verhaal in te vullen waarin nu alleen nog maar algemene woorden staan die in meerdere contexten gebruikt kunnen worden, leren de leerlingen dat wij eigenlijk veel woorden gebruiken die niet belangrijk zijn voor een specifiek verhaal. Wat sterk is, is dat de lessen eindigen met een ‘napraatje’, waarin de termen nog even worden herhaald door de leerlingen hiernaar te vragen (Kennisnet, 2016).

Bomberbot heeft geen les gemaakt die expliciet over abstract denken gaat (Bomberbot, z.d.). Om dit onderdeel van computationeel denken dus terug te kunnen vinden, moet gekeken worden naar de inhoud van alle lessen, om te zien of hier onderdelen van abstract denken in te vinden zijn. Door te abstraheren worden patronen herkend (Statter en Armoni, 2016). Hierdoor kan een stukje code korter en dus efficiënter worden gemaakt, doordat bijvoorbeeld door een for-loop te gebruiken de robot meerdere malen hetzelfde stukje patroon uitvoert (Bomberbot, z.d.). Bij Bomberbot ontvangen leerlingen meer sterren wanneer hun code efficiënter is, waardoor ze worden aangezet tot het zoeken naar (abstraheren van) patronen. Verder bevat de leerlijn een specifieke les over ‘for loops’ en over het schrijven van functies, wat ook patroonherkenning stimuleert (Bomberbot, z.d.).

In de lessen van Codekinderen ligt de focus vooral op het maken en op creativiteit (Codekinderen, z.d.). In zowel de unplugged lessen als de lessen onder de categorie ‘maken’ staat in de doelen van de docentenhandleiding niet altijd een concreet doel wat direct aan één van de aspecten van computationeel denken linkt. Zo is het doel voor de les ‘leer robottaal’ alleen ‘Leerlingen laten nadenken over een robottaal zonder dat ze daar een computer bij nodig hebben.’, terwijl in deze les duidelijk aan het in stappen denken wordt gewerkt. In de les ‘programmeer elkaar’ wordt wel in de doelen aangegeven dat de leerlingen leren wat een loop is bijvoorbeeld. Het maken van loops zorgt voor efficiëntere code. Verder wordt het

vinden van de meest efficiënte oplossing in deze leerlijn niet gestimuleerd (Codekinderen, z.d.).

5.2.2 In stukjes denken

In de leerlijn van programmeren in het PO wordt voor de onder- en middenbouw expliciet op decompositie ingegaan met twee unplugged activiteiten (Kennisnet, 2016). Voor de onderbouw zijn de bijbehorende doelen het leren een simpele activiteit in deelopdrachten op te delen en deze deelopdrachten in een logische volgorde weten te zetten. Voor de onderbouw houdt de activiteit in dat ze zaadjes gaan planten. Hiervoor dienen de leerlingen de stappen op het werkblad uit te knippen en in de juiste volgorde te zetten. Aan het einde van de les gaan de leerlingen onder andere napraten over of het in stappen verdelen de opdracht makkelijker maakt. Voor de middenbouw komt er als extra doel bij dat de leerlingen ook grote opdrachten en concrete problemen in kleine deelopdrachten weten te verdelen. De bijbehorende activiteit is het bespreken van de regels van een knikkerbaan. In de lesbeschrijving staat niets beschreven over hoe de leerlingen de verschillende aspecten van een knikkerbaan in de juiste volgorde zetten of hoe ze concrete problemen in kleine deelopdrachten weten te verdelen. Het zou kunnen dat er tijdens het maken van een knikkerbaan problemen ontstaan die door middel van kleine deelopdrachten op te lossen zijn, maar er wordt nergens in de lesbeschrijving weergegeven hoe daar mee omgegaan wordt (Kennisnet, 2016).

In vergelijking met de leerlijn ‘programmeren in het PO’ wordt bij Bomberbot minder expliciet op het ‘in stukjes denken’ ingegaan (Bomberbot, z.d.). Dit is eerder een vanzelfsprekendheid wat nodig is om de missies efficiënt te behalen. Er wordt echter wel meer ingegaan op het bedenken van functies om sub-problemen op te lossen. Het is de bedoeling dat de leerlingen de robot instructies geven om de missie te kunnen halen. Deze instructies worden opgebouwd uit verschillende pijltjestoetsen of andere taken die de robot één voor één zal uitvoeren. In de les over sequenties leren de leerlingen dat de volgorde van deze instructies invloed heeft op de uitvoering ervan. In de les over functies staat bij de doelen ook dat leerlingen leren problemen in sub-problemen op te lossen. In deze les leren de leerlingen ook een algemene functie te maken, waardoor deze vaker en op meerdere momenten toe te passen is (Bomberbot, z.d.).

Codekinderen leert niet concreet aan wat ‘in stukjes denken’ inhoudt (Codekinderen, z.d.). In de les ‘leer robottaal’ leren de leerlingen wel dat een programma werkt door het opvolgen van vele losse instructies. Door middel van een sequentie van pijltjestoetsen moeten de leerlingen een poppetje laten lopen. Ook in de les ‘lego codetaal’ staat in de doelen dat

gewerkt wordt aan het denken in stappen. Hier wordt echter niet beschreven hoe dit precies in de les aan bod komt (Codekinderen, z.d.).

5.2.3. Algoritmisch denken

Voor zowel de onderbouw, middenbouw en bovenbouw heeft programmeren in het PO lessen gemaakt om met algoritmisch denken aan de slag te gaan (Kennisnet, 2016). Al deze lessen zijn unplugged. In de kern hebben alle lessen een aantal elementen met elkaar gemeen. Elke les bevat een definitie van het woord algoritme en gebruikt op een bepaalde manier een aantal stappen om tot een antwoord te komen. Bij de onderbouw houdt dit bijvoorbeeld in dat ze leren wat de stapjes zijn die gezet moeten worden om een poppetje bij fruit te brengen en in de middenbouw leren de leerlingen stappen van instructies op te volgen, terwijl dit in de bovenbouw bijvoorbeeld gaat om stappen in de regels van een spel of het in stappen kunnen beschrijven van iets zonder dat de ander dit kan zien. In al deze lessen leren de leerlingen hoe de omschrijving van de stappen die gezet moeten worden om tot een resultaat te komen (het algoritme) belangrijk is en heel precies komt (Kennisnet, 2016).

Les 5 in de lessenserie van Bomberbot gaat over algoritmes (Bomberbot, z.d.). In deze les leren de leerlingen vanuit verschillende voorbeelden uit het dagelijks leven (tanden poetsen en een recept volgen) wat algoritmen betekenen en hoe deze efficiënter gemaakt kunnen worden. Om algoritmes efficiënter te maken herhalen ze in deze les het concept ‘for-loops’ en oefenen ze hiermee. Ze oefenen dit eerst gezamenlijk en aan het einde van de les gaan de leerlingen zelf aan de slag met de openstaande missies. In deze missies proberen de leerlingen met zo min mogelijk aanwijzingen de robot alle opdrachten uit te laten voeren, door op een efficiënte wijze het algoritme te vormen (Bomberbot, z.d.).

Codekinderen maakt gebruik van meerdere programma’s om de doelen van de lessen te behalen (Codekinderen, z.d.). Het schrijven van algoritmes is hierbij vaak belangrijk. Zo is dit bijvoorbeeld nodig om in de les ‘maak je eigen flappy bird’ het programma te laten werken. Één les voor het VO gaat over het kunnen oplossen van een Rubik’s cube. Hierin staat expliciet vermeld dat het belangrijk is dat leerlingen hiervoor het algoritme ervan begrijpen om ermee aan de slag te kunnen. Ook wordt in de unplugged lessen gebruik gemaakt van algoritmes die in het dagelijks leven te vinden zijn, zoals het volgen van een recept en het maken van een boterham met hagelslag. Echter, er wordt in de lessen voor het PO niet expliciet uitgelegd waarom een algoritme soms nodig is (Codekinderen, z.d.).

5.2.4. Evalueren

In de lessen van programmeren in het PO wordt de term ‘evalueren’ niet gebruikt (Kennisnet, 2016). Wel staan voor de onderbouw en middenbouw lessen beschreven waarin de leerlingen leren fouten in bepaalde situaties te ontdekken. Dit noemen zij ook wel ‘debuggen’ (Kennisnet, 2016). Volgens de definitie van evalueren gaat dit ook om het onderzoeken waarom een bepaalde denkstap eigenlijk is gemaakt (Hu, 2011). In deze methode wordt alleen gekeken naar welke denkstappen niet kloppen (Kennisnet, 2016). Voor de activiteit voor de onderbouw geldt dat de leerlingen een symbool moeten zoeken wat niet in de reeks past en voor de middenbouw geldt dat ze een reeks moeten overnemen en moeten debuggen wat de andere leerlingen misschien fout hadden overgenomen. Er vindt geen les plaats voor de bovenbouw over dit thema. Voor de onder- en middenbouw mist dus nog het leren evalueren van goede stappen en waarom hiervoor is gekozen (Kennisnet, 2016).

Bomberbot faciliteert evaluatie door de leerlingen zelf fouten te laten maken en hierop terug te kijken (Bomberbot, z.d.). Ook geven ze leerlingen een aantal sterren als ze een missie hebben gehaald. Het aantal sterren wat ze ontvangen is afhankelijk van de efficiëntie van de geschreven code. De leerkracht kan ook inzien hoeveel missies de leerlingen hebben weten te behalen en hoeveel sterren hierop gehaald zijn. Toch zou Bomberbot volgens de ‘pedagogical evaluation of Bomberbot’ (persoonlijke communicatie) nog meer op zelfevaluatie kunnen focussen, door bijvoorbeeld aan te geven waar een leerling een fout maakt of hoeveel pogingen een leerling nodig had. Ook kan dan worden gekeken naar hoe de leerling vooruit is gegaan in de verschillende pogingen (Bomberbot, z.d.).

De verschillende lesactiviteiten van Codekinderen maken gebruik van verschillende programma’s en aanpakken (Codekinderen, z.d.). Ook is er geen sprake van een vaste volgorde waarin de leerlingen de activiteiten moeten doorlopen. Daarom sluiten niet alle activiteiten even goed op elkaar aan en kunnen de leerlingen minder goed in de gaten houden hoe hun leerproces gaat. Er is geen volgsysteem. Wel wordt bijvoorbeeld in de les ‘programmeer elkaar’ genoemd dat het belangrijk is dat de leerlingen leren ‘debuggen’ als het ze niet lukt. Er wordt echter niet omschreven hoe dit ‘debuggen’ precies in zijn werk gaat. Ook vindt in de lessen geen evaluatief eindgesprek plaats (Codekinderen, z.d.).

5.2.5. Generaliseren

In programmeren in het PO staan geen lessen opgesteld waarin generaliserend denken voorop staat (Kennisnet, 2016). Echter, het principe van generaliseren is wel terug te vinden in onderdelen van meerdere lessen. Zo moeten de kleuters bijvoorbeeld in de lessen van patroonherkenning patronen zien te vinden in het schoolgebouw, waardoor ze erachter zullen

komen dat patronen niet alleen online belangrijk zijn en dus in meerdere situaties toepasbaar zijn (Selby & Woollard, 2013). Doordat programmeren in het PO veelal gebruik maakt van unplugged lessen, worden op deze manier de onderdelen van programmeren in vele dagelijkse activiteiten herkend, van het zaaien van een plantje tot het volgen van spelregels (Kennisnet, 2016). Voorwaarde voor het leren generaliseren van concepten is wel dat de leerlingen een concept meerdere malen zien terugkomen, waardoor duidelijk wordt dat dit in meerdere contexten toepasbaar is (Selby en Woollard, 2013). In deze methode staat deze voorwaarde niet beschreven (Kennisnet, 2016). Wel staat een aantal keer in de doelen van een les beschreven dat leerlingen leren inzien dat situaties uit het dagelijks leven te vertalen zijn naar online situaties en andersom (Kennisnet, 2016).

In veel Bomberbot lessen worden voorbeelden uit de praktijk aangehaald, om bepaalde concepten uit te leggen (Bomberbot, z.d.). Hierdoor wordt duidelijk dat de concepten niet alleen voor programmeren, maar ook in andere contexten nuttig zijn en dus gegeneraliseerd kunnen worden. Ook gaan de leerlingen in een tweede jaar Bomberbot beginnen met een stukje automatisering, door recursieve functies te maken. Verder wordt veel ingegaan op het veralgemeniseren van stukjes code, door te werken aan efficiëntie en het schrijven van extra functies die vervolgens kunnen worden hergebruikt (Bomberbot, z.d.).

Codekinderen heeft veel lesactiviteiten ingedeeld onder de categorie ‘unplugged’ (Codekinderen, z.d.). Deze activiteiten behoeven geen computer en zijn daarom uitermate geschikt om uit het dagelijks leven te putten. Echter, veel lessen zijn niet op deze manier opgebouwd en zijn vooral fysieke versies van problemen die ook op een computer opgelost hadden kunnen worden. De lessen ‘in en output met muffins’ en ‘de hagelslagrobot’ zijn wel terug te vinden in het dagelijks leven, aangezien ze hier leren dat recepten volgen en de stappen die je neemt om iets te maken ook in het dagelijks leven erg precies moeten zijn (Codekinderen, z.d.).

5.3 Methoden en leertheorieën

Wanneer de methoden met elkaar worden vergeleken, blijkt dat over het algemeen Bomberbot de meeste pluspunten heeft en Codekinderen de meeste minpunten. Programmeren in het PO benoemt het meest expliciet de verschillende aspecten van computationeel denken, maar hier wordt in de lessen niet altijd diep op ingegaan. Bomberbot benoemt de concepten minder, maar heeft de meest zichtbare lijn in het leren en verwacht qua niveau ook meer van de leerlingen. Codekinderen heeft geen duidelijke leerlijn en beschrijft ook niet duidelijk wat het niveau van de lessen is. Wel maakt het gebruik van veel verschillende programma’s, waardoor de lessen divers zijn.

Volgens Piaget (1964) zitten de meeste leerlingen wanneer ze deze lessen krijgen in de concreet operationele fase (in de midden- en bovenbouw van de basisschool), wat inhoudt dat ze objecten binnen bepaalde contexten kunnen plaatsen. Pas later leren ze abstracter, idealistischer en logisch denken. Dit kan betekenen dat bepaalde onderwerpen nog lastig zullen zijn voor de leerlingen. Zo zou het schrijven van abstracte code, wat bijvoorbeeld in Bomberbot steeds moeilijker wordt, nog lastig kunnen zijn voor de leerlingen. Echter, volgens Bain (2004) zorgt het stellen van hoge verwachtingen voor de leerlingen juist voor uitdagingen, wat ze kan stimuleren sneller de volgende fase van Piaget in te gaan.

Ook omgevingsfactoren, interactie en genen hebben invloed op hoe deze methoden bij de leerlingen aanslaan, maar hierover kan van de verschillende methoden niets worden gezegd, omdat dit uit de verschillende bronnen niet te achterhalen is. Wel kan worden gekeken naar op welke manier de methoden feedback geven aan de leerlingen. Opvallend is dat Bomberbot zowel op taak- (is de missie geslaagd) als procesniveau (hoe efficiënt is de taak geslaagd) feedback geeft. Feedback op zelfregulerend- en op persoonsniveau is in geen van de methoden zichtbaar. Om feedback op zelfregulerend niveau aan de lessen toe te voegen, kunnen de leerkrachten bijvoorbeeld met de leerlingen in gesprek gaan over de manier waarop ze iets hebben aangepakt (Bakx, 2015). Programmeren in het PO beschrijft al wel veel vragen voor de gesprekken tussendoor en aan het einde van opdrachten, maar dit kan nog specifieker over zelfregulatie gaan.

Het aspect relatie van de zelfdeterminatietheorie (Bakx, 2015) krijgt in geen van de methoden aandacht. Dit is daarom ook afhankelijk van wat de leerkracht buiten de lesvoorbereidingen om doet. Al zou het niveau van Bomberbot soms (te) hoog kunnen zijn, krijgen de leerlingen hierin wel de meeste autonomie over hun leerproces, omdat ze veel zelfstandig kunnen werken. Doordat de missies op meerdere niveaus te halen zijn kunnen ze zich ook competent voelen. Codekinderen werkt steeds met verschillende methoden, waardoor de leerlingen veelal afhankelijk zullen zijn van de leerkracht aangezien ze steeds opnieuw moeten uitzoeken hoe iets werkt. Hier hebben de kinderen dus minder autonomie. Verder valt bij Codekinderen op dat veel lessen gefocust zijn op hoe je iets kunt maken. De methode is daarom erg handelingsgericht (Bakx, Ros en Bolhuis, 2017). Vooral Bomberbot is daarentegen meer opbrengstgericht, omdat een van de doelen is dat de leerlingen de missies halen. Door goed naar de specifieke klas te kijken en op basis daarvan ook handelingsgericht te werken, zal dit onderwijs aansluiten bij de leerlingen (Bakx, Ros en Bolhuis, 2017).

6. Discussie

De hypothese voor deze scriptie was dat het antwoord op de onderzoeksvraag genuanceerd zou zijn, doordat waarschijnlijk niet elke methode in dezelfde mate aandacht zou besteden aan alle eigenschappen van computationeel denken, maar het zou kunnen dat dit niet gelijk op gaat met in hoeverre de methoden iets vanuit het oogpunt van cognitieve leertheorieën hebben aangepakt.

Uit de resultaten blijkt dat bij elke methode bij elk aspect van computationeel denken iets te zeggen valt over hoe dit wordt toegepast. Wel zit er verschil in de manier waarop dit wordt toegepast. In grote lijnen wordt duidelijk dat in de methode ‘programmeren in het PO’ expliciet wordt ingegaan op het aanleren van de technische componenten van computationeel denken, maar het evalueren en generaliseren niet op deze manier aan bod komt. Wel halen zij veel voorbeelden uit de praktijk aan, ook om de concepten samen in de les te bespreken. Het is niet geheel duidelijk of dit geldt voor alle groepen, aangezien soms lessen over bepaalde concepten missen. Bomberbot heeft daarentegen wel erg duidelijk een leerlijn opgesteld voor alle groepen op de basisschool. In deze methode vindt niet elk aspect van computationeel denken expliciet plaats in de leerlijn, maar was dit wel zichtbaar als dieper op de lessen in werd gezoomd. Het sterke punt aan Bomberbot is dat de focus flink ligt op het efficiënt maken van code, waardoor in dit proces meteen gewerkt wordt aan alle verschillende aspecten van computationeel denken. Ook zijn de leerlingen eigenaar van hun eigen leerproces, doordat ze veel zelf oefenen en er een duidelijke opbouw in de lessen is. Wel zou de methode aan het einde van de lessen nog uitgebreider kunnen evalueren, om de leerlingen te laten inzien wat ze die les precies hebben geleerd. In codekinderen wordt regelmatig niet diep genoeg ingegaan op de aspecten van computationeel denken. Deze methode is erg gefocust op het maken van creatieve lessen, waardoor het erg sterk is in het laten inzien dat de aspecten van computationeel denken ook in de praktijk erg bruikbaar zijn. Echter, hierin mist nog wel het benoemen van alle aspecten die hierbij komen kijken en het vinden van uitdaging in het nadenken over de probleemstelling. De methode maakt gebruik van meerdere programma’s waaromheen verschillende lessen zijn gevormd. Tussen deze lessen vindt geen concrete opbouw plaats, waardoor de leerkracht zelf moet bepalen of een klas een les al zou begrijpen. Hierdoor mist soms voorkennis en diepgang.

Deze analyse was kwalitatief en is veelal gebaseerd op documenten die online te vinden waren of persoonlijk zijn toegestuurd. Verder is de inhoud van de beschikbare lessen bekeken. Toch is de conclusie beïnvloed door mijn interpretatie van de methoden. Om een

genuanceerder beeld te vormen van de verschillende methoden zou het daarom verstandig zijn deze analyse door meerdere studenten of wetenschappers uit te laten voeren, de resultaten met elkaar te vergelijken en op die manier tot een gezamenlijk standpunt te komen. Verder was het lastig om deze methoden met elkaar te vergelijken, omdat ze allemaal op een andere manier zijn opgebouwd. Vooral bij de methoden programmeren in het PO en codekinderen is de daadwerkelijke invloed ervan afhankelijk van de leerkracht, aangezien hij/zij de opbouw in de lessen moet bepalen. Voor een school is het daarom ook erg lastig om te kiezen tussen deze verschillende methoden. Het zou nuttig zijn een vervolgonderzoek plaats te laten vinden, waarin door meerdere mensen nog meer methoden meegenomen worden in een analyse op aspecten van computationeel denken. Dan kunnen namelijk precies de verwachtingen van leerkrachten en de impact op het gebied van computationeel denken (en misschien zelfs ook de andere onderdelen van digitale geletterdheid) onder elkaar worden gezet, voor een goed overzicht om uit te kiezen. Dat zou het starten aan een methode om computationeel denken in het onderwijs te stimuleren flink vergemakkelijken.

7. Conclusie

De onderzoeksvraag van deze scriptie was: “Welke eigenschappen van computationeel denken zijn belangrijk in het onderwijs en hoe worden deze door de huidige methoden geïmplementeerd wanneer gekeken wordt naar de beschreven methode?”. Er werd verwacht dat het antwoord op deze vraag genuanceerd zou zijn, doordat de definitie van computationeel denken niet eenduidig is, de methoden hierdoor verschillende aanpakken zullen hebben en ook cognitieve leertheorieën invloed hebben op de manier waarop de methoden bij de leerlingen aankomen.

Uit een vergelijking van verschillende definities van computationeel denken bleek dat de meta-analyse van Selby en Woollard (2013) belangrijke eigenschappen noemde, namelijk: abstract denken, in stukjes denken, algoritmisch denken, evalueren en generaliseren.

Voor het beantwoorden van het tweede gedeelte van de onderzoeksvraag werden de methoden ‘Programmeren in het PO’, ‘Bomberbot’ en ‘Codekinderen’ geanalyseerd, waarin gekeken werd naar deze vijf eigenschappen. Hieruit kan worden geconcludeerd dat al deze methoden werken aan computationeel denken in het basisonderwijs. Uit de tabel blijkt dat Bomberbot de meeste pluspunten en Codekinderen de meeste minpunten heeft. Als kwantitatief naar deze methoden zou worden gekeken, zou Bomberbot dan ook het beste uit de verf komen. Echter, nu kwalitatief gekeken is naar de precieze inhoud van de methoden, werd duidelijk dat programmeren in het PO de verschillende aspecten van computationeel denken het meest expliciet benoemt, Bomberbot de meest duidelijke leerlijn heeft en een hoger denkniveau van de leerlingen verwacht en de lessen van Codekinderen het meest divers zijn. Doordat niet duidelijk is of de eigenschappen van computationeel denken allemaal even belangrijk zijn, zijn ze niet tegen elkaar op te wegen. Ook uit de analyse van de verschillende cognitieve leertheorieën blijkt dat niet gelijk duidelijk wordt welke methode de leerlingen het beste bereikt. Het antwoord op het tweede deel van de onderzoeksvraag is daarom ook genuanceerd; het is afhankelijk van de klas en leerkracht welke methode het beste computationeel denken aan die leerlingen leert.

In conclusie, de hypothese dat het antwoord op de onderzoeksvraag genuanceerd zou zijn klopt. Wel blijkt over het algemeen dat de verschillende methoden allemaal computationeel denken aanleren. Als in het onderwijs met deze methoden wordt gewerkt, zal dit voor de toekomst dus betekenen dat we problemen systematischer kunnen oplossen, kritischer naar de resultaten van computers kunnen kijken en beter zullen begrijpen waar

kunstmatige intelligentie ons kan helpen met (technische) problemen. Een leuke investering voor nu, een gouden kans voor de toekomst van kunstmatige intelligentie!

8. Literatuur

Bakx, A. (2015). *De pedagogisch sensitieve leraar.* Coutinho.

Bakx, A., Ros, A., & Bolhuis, E. (2017). *Cyclisch onderwijs ontwerpen* (2de herz. ed.). Coutinho.

Bain, K. (2004). *What the best college teachers do.* Harvard University Press.

Bomberbot (z.d.). Programmeren voor groep 5 t/m 8 [Lessen]. Geraadpleegd van https://leerkracht.bomberbot.com/app/curriculum/f0bcbea0-60d8-4898-b376-5083f2c93bbe

Bosse, T. (2019). Sociale kunstmatige intelligentie [Oratie]. Geraadpleegd van https://repository.ubn.ru.nl/bitstream/handle/2066/200202/200202.pdf?sequence=1&isAllowed=y

Codekinderen. (z.d.). *Welkom*. Wikiwijs. https://maken.wikiwijs.nl/100525/CodeKinderen

Gadanidis, G. (2016). Artificial intelligence, computational thinking, and mathematics education. *The International Journal of Information and Learning Technology*, 133-139. doi:10.1109/TALE.2016.7851769

Hu, C. (2011). Computational thinking: what it might mean and what we might do about it. *ITiCSE*, 223-227. doi:10.1145/1999747.1999811

Kennisnet. (2019). *Computational thinking?* Wikiwijs. https://maken.wikiwijs.nl/70012/Workshop_Computational_thinking#!page-1652296

Kennisnet. (2016, 26 mei). *Programmeren in het PO*. Wikiwijs. https://maken.wikiwijs.nl/74282/Programmeren_in_het_PO

Kijk op ontwikkeling. (2014, 23 november). *Wat is cognitieve ontwikkeling?* https://www.kijkopontwikkeling.nl/artikel/wat-cognitieve-ontwikkeling/

KNAW. (2012). Digitale geletterdheid in het voortgezet onderwijs. *Koninklijke Nederlandse Akademie van Wetenschappen*, 12-22. doi:20121027

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. doi: 10.1145/1929887.1929902

Piaget, J. (1964). Cognitive development in children: Piaget. *Journal of research in science teaching*, *2*(3), 176-186. doi:10.1002/tea.3660020306

Poland, M., van Oers, H. J. M., & Terwel, J. (2010). Effecten van het leren schematiseren van jonge leerlingen op hun latere prestaties in het reken-wiskundeonderwijs. *Pedagogische studiën*, *87*(2), 105-117.

Selby, C. C., & Woollard, J. (2013, 1-3 juli). *Computational thinking: the developing definition.* ITiCSE Conference 2013, University of Kent, Canterbury, Engeland. https://www.researchgate.net/publication/299450690_Computational_thinking_the_developing_definition

SLO. (2020a, 6 januari). *Computational thinking*. https://www.slo.nl/vakportalen/vakportaal-digitale-geletterdheid/computational-thinking/

SLO. (2020b, 13 februari). *Achtergrond digitale geletterdheid*. https://www.slo.nl/vakportalen/vakportaal-digitale-geletterdheid/achtergrond/

Statter, D., & Armoni, M. (2016). Teaching Abstract Thinking in Introduction to Computer Science for 7th Graders. *Proceedings of the 11th Workshop in Primary and Secondary Computing Education.* 80-83. doi:10.1145/2978249.2978261

Thijs, A., Fisser, P., & Hoeven, M. van der (2014). *21e eeuwse vaardigheden in het curriculum van het funderend onderwijs*. Enschede: SLO.

Ubiquity. (2007). An Interview with Peter Denning on the great principles of computing. *Ubiquity*, 1. doi:10.1145/1276162.1276163

Van Ramele, T. (2019). *Digitale geletterdheid in het onderwijs* [Bachelorscriptie, Saxion]. (Hbokennisbank).

Voogt, J., Brand-Gruwel, S., & van Strien, J. (2017). *Effecten van programmeeronderwijs op computational thinking.* NRO. https://www.nro.nl/sites/nro/files/migrate/003-Antwoord-Rapport-Programmeeronderwijs.pdf

Wing, J.M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35. doi:10.1145/1118178.1118215

Zeng, D. (2013). From Computational Thinking to AI Thinking [Een brief van de editor]. *IEEE Intelligent Systems*, *28*(06), 2-4.