Computational thinking

Een manier om probleemoplossend vermogen te vergroten

Gerard Dummer, Pabo Hogeschool Utrecht

Inleiding

Hoe goed begrijp je de wereld om je heen nog? Welke invloed wil je kunnen uitoefenen op de wereld om je heen? Zie je ook de mogelijkheden die de wereld om je heen biedt voor het onderwijs? Het rapport *De Toekomst Telt* (Boswinkel & Schram, 2011) noemt de wereld een *black box world* door alle digitale apparaten die we wel gebruiken maar niet meer begrijpen. Honegger (2015) noemt het daarom een filosofisch uitgangspunt als hij zegt dat er aandacht moet zijn voor computer *sciences* in het onderwijs, zodat kinderen beter in staat zijn om de wereld om zich heen te begrijpen en naar hun hand te zetten. Verder laat Van Keulen (2010) in zijn publicatie *Wetenschap en Techniek* zien hoeveel mogelijkheden er in het dagelijks leven van kinderen zijn om bij aan te sluiten.

Dit artikel gaat over *computational thinking*. Uitgangspunt daarbij is dat computational thinking helpt om de wereld om je heen beter te begrijpen en naar je hand te zetten. Ik leg uit wat computational thinking is en hoe je hier mee op de basisschool aan de slag kunt gaan.

Wat is computational thinking?

In haar veelgeciteerde artikel *Computational Thinking* legt Wing (2006) uit wat computational thinking (CT) is. CT gaat over het oplossen van problemen. Preciezer gezegd: de vaardigheid om moeilijke problemen zo te herformuleren dat we ze kunnen oplossen. Het gaat daarbij niet zo zeer om programmeren, maar om conceptueel kunnen denken. Wing geeft aan dat CT een fundamentele vaardigheid is en niet iets mechanisch. Het is een menselijke manier van denken om problemen op te lossen. Het gebruikt kennis en vaardigheden die voortkomen uit het wiskundig en het technisch denken. Het gaat daarbij niet om de software en hardware tools maar om de concepten. Het is voor iedereen omdat het verweven is in ons dagelijks leven. CT gaat helpen, zegt Wing, bij het oplossen van huidige en toekomstige (wetenschappelijke) problemen.

Concepten en aanpakken CT / TPACK en CT

Met deze omschrijvingen kun je nog niet aan de slag in het onderwijs natuurlijk. Gelukkig zijn er al verschillende uitwerkingen gemaakt die houvast bieden. Deze uitwerkingen geven aan welke concepten belangrijk zijn en welke aanpakken hierbij passen (Brennan, Balch & Chung, 2015; Curzon, Dorling, Ng, Selby & Woollard, 2014; Programmeren in het PO, 2016) . Ik koppel deze concepten en aanpakken aan het TPACK-model (Koehler & Mishra, 2006). Het TPACK-model zegt dat je als leer-

kracht in staat moet zijn om vakinhouden over te dragen met bijpassende didactieken en technologieën. Daarbij wordt technologie ruim opgevat; van pen tot computer. Dit

Tabel 1. Computational thinking.

Computational Thinking			
Computational Concepts (Vakinhouden)	**Computational Practices (Didactiek)**	**Computational Technology (Technologie)**	**Computational Perspectives (Context)**
• Algoritme • Herhaling • Patronen • Abstractie • Parallelisme • Decompositie • Logica • Data • Voorwaarden • Functies • Representaties • Operators	• Experimenteren en verbeteren • Knutselen • Testen en debuggen • *Reusing* en *remixing* • Ontwerpen • Samenwerken • Evalueren • Programmeren	• *Unplugged* • Technische hardware • *Plugged*	• Doorzettings-vermogen stimu-leren • Uitbreiding van expressiemiddelen • De kracht van samenwerken ont-dekken • Zelfverzekerder over de wereld om hen heen

Tabel 2. Omschrijving van computational concepts.

Computational Concepts	**Omschrijving**
Algoritme	Instructies die je na elkaar moet uitvoeren. Een verzameling regels. Effectieve en efficiënte algoritmes besparen tijd en geld.
Herhaling	Een algoritme meerdere keren doorlopen.
Patronen	Patronen helpen om voorspellingen te maken, regels vast te stellen en algemenere problemen op te lossen. Door te generaliseren kun je een bepaald probleem op dezelfde manier oplossen.
Abstractie	Vaststellen wat belangrijk is en wat weggelaten kan worden zodat je com-plexe problemen kunt oplossen.
Parallelisme	Twee of meer algoritmes tegelijkertijd uitvoeren.
Decompositie	Een complex probleem in kleinere stukken opknippen zodat het probleem hanteerbaarder wordt.
Logica	Uit kunnen leggen waarom iets gaat zoals het gaat. Of een manier om uit te leggen waarom iets niet gaat zoals het zou moeten gaan.
Data	Invoeren, bewaren, ophalen en vernieuwen van waarden. Ook wel varia-bele genoemd. Bijvoorbeeld de variabele score die wordt bewaard, ge-toond en verhoogd of verlaagd. Of een lijst met namen waaruit gekozen kan worden.
Voorwaarden	Een besluit nemen gebaseerd op een bepaald aantal voorwaarden. Het gaat hierbij om “als-dan-anders”-redeneringen. Ook wel ‘selection’ genoemd.
Functies	Een ‘hulp’programma binnen het ‘hoofd’programma dat hergebruikt kan worden. Wordt ook wel sub routine genoemd.
Representaties	Het weergeven en organiseren van gegevens in passende grafieken, lijsten, teksten of plaatjes.
Operators	Operators zijn hulpmiddelen voor wiskundige, logische en string uitdrukkingen. Voorbeelden zijn +, -, *, EN, OF, NIET.

alles vindt plaats in een bepaalde context. De vakinhouden van CT noemen we *computational concepts.* De didactieken van CT heten *computational practices.* De technologieën van CT kun je onderverdelen in *unplugged* (zonder computer maar met huis-, tuin- en keukenmiddelen), technische hardware (specifieke didactische

materialen) en *plugged* technologie (software, apps en websites). Onder de context schaar ik onderwerpen die Brennan, Balch en Chung (2015) onder *computational perspectives* scharen. Het zijn vooral houdingsaspecten die bij CT komen kijken. In tabel 1 heb ik de *concepts*, *practices,* technologieën en *perspectives* op een rijtje gezet. In tabel 2 licht ik de *computational concepts* nog verder toe.
Van de *computational concepts* werk ik er één uit met concrete voorbeelden: algoritmes. Ook van *compational practices* ga ik voor dit hoofdstuk in op één onderwerp: *reusing en remixing*. Tot slot sta ik stil bij de verschillende technologieën die je kunt gebruiken.

Computational concept: algoritme

Er zijn heel veel soorten algoritmes. Kinderen leren al verschillende algoritmes op school bij bijvoorbeeld rekenen (een stappenplan gebruiken om een deling op te lossen) en spelling (een stappenplan voor het bepalen van de d's en dt's). Maar ze komen ook algoritmes tegen in het dagelijks leven en algoritmes waar computerprogramma's gebruik van maken. Algoritmes helpen om een bepaald 'probleem' op te lossen. Ik geef een aantal voorbeelden hoe je met verschillende algoritmes aan de slag kunt gaan.

Voorbeeld 1: Hagelslagrobot

De activiteit van de hagelslagrobot of sandwichrobot is een mooie activiteit om kinderen te leren precies instructies te formuleren. De leraar speelt een robot die een broodje hagelslag klaar moet maken. De leraar heeft alle materialen voor het maken van een broodje hagelslag klaarliggen (brood in broodzak, boter, mes, bord, pak hagelslag). De leerlingen moeten bedenken welke instructies ze de leerkracht moeten geven om ervoor te zorgen dat er een gesmeerd broodje hagelslag komt te liggen. De leerlingen mogen daarbij alleen woorden gebruiken die zijn voorgeschreven. Is de instructie niet helder genoeg (slaan ze een stap over) dan loopt de robot vast en moeten de instructies preciezer worden geformuleerd.

http://code-it.co.uk/unplugged/jamsandwich http://www.codekinderen.nl/leerling/unplugged/sandwich-robot/

De hageslagrobot is een mooie aanleiding om het met kinderen te hebben over apparaten in het huis. Hoe zou het algoritme van bijvoorbeeld de wasmachine eruitzien? Een wasmachine weet natuurlijk niet of de was schoon is maar het draait zijn programma af. Hoe zou je de instructies voor de wasmachine kunnen opschrijven?

Voorbeeld 2: Sorteeralgoritme

Hoe zorg je ervoor dat je klasgenootjes met zo min mogelijk instructies op lengte of leeftijd gaan staan? Als je dat zo efficiënt mogelijk wilt doen dan denk je na over een sorteeralgoritme. Een mooi uitgewerkt voorbeeld vind je op CS *Unplugged* over het sorteren van fotorolhoudertjes met verschillende gewichten. Hoe kun je die sorteren

in zo min mogelijk stappen? Neem je eerst de grootste, zwaarste of langste en vergelijk je die met de rest? Of kies je een willekeurige uit en vergelijk je de volgende daarmee? Of splits je de groep en sorteer je eerst de groepjes?

http://csunplugged.org/sorting-algorithms/

De opdracht met het sorteeralgoritme is een mooie aanleiding om het met kinderen te hebben over de manier waarop de computer bepaalt hoe die iets op alfabetische volgorde kan zetten (bijvoorbeeld in de Verkenner of in Excel).

Voorbeeld 3: Zoekalgoritme

Hoe vind je zo snel mogelijk het getal dat de meester of juf in zijn hoofd heeft? Als je op zoek bent naar een item in een grote verzameling dan ben je bezig met het maken van een zoekalgoritme. Een handige manier om snel dat getal te vinden dat de meester of juf in gedachte heeft is door steeds de helft te nemen van het totale aantal (binair zoeken). Als je aan een getal denkt onder de honderd kun je vragen of het getal groter of kleiner is dan vijftig. Is het groter dan vijftig dan heb je al de helft van de getallen uitgeschakeld. De volgende vraag is dan of het getal groter of kleiner is dan vijfenzeventig. En zo verder tot je het getal gevonden hebt.

http://csunplugged.org/searching-algorithms/

De opdracht met het zoekalgoritme is een mooi gelegenheid om het over de zoekmachines te hebben. Hoe vinden zij in hun database de pagina die voor jou het meest waardevol is?

Computational Practices: reusing en remixing

Een *computational practice* die Brennan, Balch en Chung (2015) noemen is *reusing* en *remixing*: hergebruik van en voortbouwen op bestaande materialen. Je hoef niet zelf het wiel uit te vinden, je kunt gebruiken maken van wat er al is. Zij betrekken dit op het programma *Scratch* van MIT. Scratch is een visuele programmeeromgeving waarin je blokjes aan elkaar sleept om een programma te schrijven. Scratch is een online programma waarin het makkelijk is om elkaars projecten te hergebruiken en daar op voort te bouwen. Dat is een houding die voortkomt uit de *open source* programmeerwereld. Daarin worden programmacodes met elkaar gedeeld. Bijvoorbeeld op een site zoals *GitHub*

https://github.com/

Hergebruik en voortbouwen zie je niet alleen op het gebied van programmeren maar ook op gebied van foto's, video's en teksten. Kinderen daar bewust mee in aanraking laten komen is een waardevolle manier van werken. Rondom programmeren kan dat bijvoorbeeld door in *Scratch* verder te bouwen op het project van iemand anders. Omdat zo'n project een mooi uitgangspunt kan zijn voor je eigen project of omdat je

kunt leren van de codes van anderen. De makers van *Scratch* hebben zelf al een aantal startprojecten gemaakt die je kunt aanpassen om te oefenen met *reusing* en *remixing.*

https://scratch.mit.edu/projects/10014986/
https://scratch.mit.edu/projects/10128431/
https://scratch.mit.edu/projects/22162012/

Als je verder wilt gaan met een project van iemand anders, kan daar geen copyright op zitten. Dit biedt dan ook een mooie gelegenheid om het met leerlingen te hebben over auteursrecht. Specifiek gaat het daarbij om *Creative Commons*, een licentievorm die ervoor zorgt dat je onder bepaalde voorwaarden het werk van een ander mag gebruiken. Zulke voorwaarden zijn bijvoorbeeld dat je aan moet geven wie de oorspronkelijke auteur is en dat je jouw nieuwe werk onder dezelfde voorwaarden ook weer vrij moet geven.

Reusing en *remixing* gebruiken als werkvorm opent ook de deur naar een andere manier van werken. Eentje waarin het normaal is dat je online samenwerkt met anderen, omdat je ervaart dat die samenwerking meer op kan leveren dan alleen werken. Het beste voorbeeld daarvan is Wikipedia, het grootste samenwerkingsproject op internet bedoeld om alle mensen toegang te geven tot alle kennis.

Tot slot: Computational Technology

Welke middelen kun je nu inzetten om met kinderen te werken aan *computational concepts*? Hierbij kun je weer het eerdergenoemde onderscheid maken in *unplugged* activiteiten, activiteiten met technische hardware en *plugged* activiteiten.
De hagelslagrobot, het sorteeralgoritme en het zoekalgoritme zijn voorbeelden van *unplugged* activiteiten. Meer voorbeelden van *unplugged* activiteiten vind je op de website *CSUnplugged*
Onder technische hardware versta ik de middelen waarbij je niet achter het computerscherm hoeft te kruipen maar die wel gemaakt zijn om *computational concepts* uit te leggen. Er zijn al veel voorbeelden op de markt hiervoor. Zo is er voor de jongsten het bordspel *RobotTurtle* waarin je codekaartjes moet leggen om de robot over een spelbord te kunnen verplaatsen. Ook voor jonge kinderen is de *Beebot*, een robotbijtje die je eenvoudig kunt programmeren op de bij zelf. *Cubetto* is ook een robotje voor jonge kinderen waarbij je de code op een apart bord achter elkaar kunt plaatsen.
Voor *plugged* activiteiten ga je achter je computerscherm zitten om te programmeren. Ik noemde al *Scratch*, voor de bovenbouw. Voor jongere kinderen (groep 3 of 4) is er ook *ScratchJR* voor op de tablet. Daarnaast kun je terecht op verschillende websites zoals *code.org* en *bomberbot*. Ook zijn er allerlei apps voor op je tablet zoals *LightBot* en *Hopscotch.*

http://csunplugged.org/

http://www.robotturtles.com/
http://www.codekinderen.nl/leerling/programmeren/bee-bot/
https://www.primotoys.com/
https://scratch.mit.edu/
https://www.scratchjr.org/index.html
https://lightbot.com/
https://www.gethopscotch.com/
https://www.khanacademy.org/partner-content/pixar

Literatuur

Boswinkel, N., Schram, E. (2011). *De Toekomst Telt*. Enschede: SLO.

Brennan, K. Balch, C. & Chung, M. (2015). Creative Computing. Geraadpleegd op 1 oktober 2016, van http://scratched.gse.harvard.edu/guide/.

Curzon, P. Dorling, M., Ng, T. Selby, C. & Woollard, J. (2014). Developing computational thinking in the classroom: a framework. Geraadpleegd op 1 oktober 2016 van https://academy.bcs.org/sites/academy.bcs.org/files/DevelopingComputationalThinkingInTheClassroomaFramework.pdf.

Honegger, B.D. (2015). We are all excited - but why? Geraadpleegd op 1 oktober 2016, van http://doebe.li/talks/scratch15/index.html.

Keulen, H. Van. (2010). *Wetenschap en techniek. IJkpunten voor een domein in ontwikkeling*. Den Haag: Platform BetaTechniek.

Koehler, M.J., & Mishra, P. (2006). What Is Technological Pedagogical Content Knowledge? Geraadpleegd op 1 oktober 2016, van http://www.citejournal.org/wp-content/uploads/2016/04/v9i1general1.pdf.

Programmeren in het PO. (2016). *Programmeren in het PO.* Geraadpleegd op 1 oktober 2016, van http://maken.wikiwijs.nl/74282/Programmeren_in_het_PO.

Wing. J.M. (2006). Computational Thinking. Geraadpleegd op 1 oktober 2016, van https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf.

Dummer, G. (2017). Computational thinking. Een manier om probleemoplossend vermogen te vergroten. In: M. van Zanten (red.). *Rekenen-wiskunde in de 21e eeuw. Ideeën en achtergronden voor primair onderwijs* (pp. 131-136). Utrecht / Enschede: Panama, Universiteit Utrecht / NVORWO / SLO.