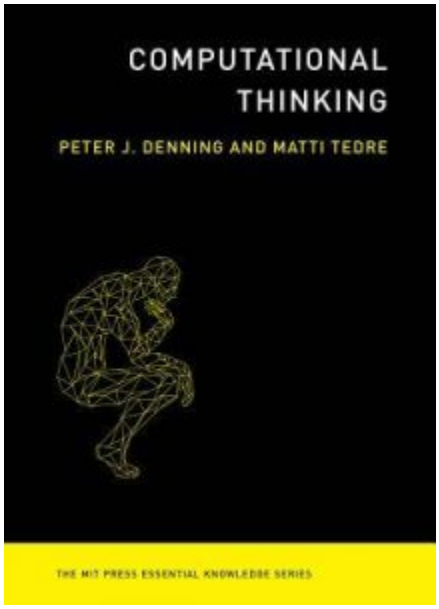
Gelezen: Computational Thinking (Peter J. Denning en Matti Tedre)

Auteur samenvatting: Pierre Gorissen, gepubliceerd op 9-1-2020 via https://kortelink.com/2QCfSdO



Het is het onderdeel van de competenties rond ict-geletterdheid waar docenten, leraren, eigenlijk bijna iedereen het meest moeite mee hebben: computational thinking.

Al was het maar omdat het in tegenstelling tot de andere onderdelen (ict-basisvaardigheden, informatievaardigheden, mediawijsheid) een Engelstalig begrip is. Hele discussies vinden plaats om na te denken over een goede Nederlandse vertaling. Maar wat is het nou eigenlijk, waar komt het vandaan, waarom moeten we er aandacht aan besteden?

Het boek "Computational Thinking" van Peter Denning en Matti Tedre probeert antwoord te geven op deze vragen. Daar hebben ze bijna 200 pagina's voor nodig (150 zonder d[1]e bijlagen). In het Engels. Dus daarom een "korte" samenvatting van de inhoud. Niet met de bedoeling om die 150 pagina's te vervangen, maar om je een beeld te geven van de inhoud.

Introductie

De auteurs beginnen met een stapje terug en de vragen die mensen hebben bij de verregaande digitalisering van de maatschappij. Ze stellen zichzelf als doel om met het boek te helpen bij het innemen van een beter onderbouwde positie in het gesprek daarover. Daarbij beginnen ze met de relativering: dat veel van ons dagelijks leven beïnvloedt wordt door computers is niet nieuw. Ook voorgaande technologieën hadden zulke ingrijpende gevolgen voor de samenleving.

Definitie van Computational Thinking (CT)

Uiteraard geven de auteurs ook een/hun definitie van het begrip computational thinking (CT). In het Engels: *Computational thinking is the mental skills and practices for*

- *designing computations that get computers to do jobs for us, and*
- *explaining and interpreting the world as a complex of information processes.*

[1] https://mitpress.mit.edu/books/computational-thinking

In het Nederlands: Computational Thinking (*CT) zijn de mentale vaardigheden en praktijken voor het*

- ***ontwerpen** van berekeningen waarmee computers werk voor ons kunnen doen, en*
- ***verklaren** en interpreteren van de wereld als een geheel van informatieprocessen.*

Die definitie vergt ook in het boek enige uitleg. Bij berekeningen gaat het zowel om eenvoudige wiskundige berekeningen, maar ook complexe zoals het voorspellen van het weer. Computers zijn die zware kasten die de meeste van ons alleen van vroeger kennen, je laptop, maar ook je telefoon, je horloge, delen van je auto etc.
Bij werk betreft het taken die waardevol zijn (niet persé een baan), zeker niet alleen eenvoudige taken. Niet alles is te automatiseren en in veel gevallen is er behoefte aan zowel domeinkennis als CT-kennis.
De definitie laat ook de twee kanten van CT zien die de auteurs hanteren: het ontwerpen enerzijds, maar ook het gebruik ervan ten behoeve van het verklaren en interpreteren van de wereld. Ze komen daar op terug als het gaat om de integratie van ict/automatisering in de verschillende reeds bestaande wetenschappen.

De auteurs maken daarnaast onderscheid tussen “basis CT” als ze het hebben over CT zoals dat in de regel in het basis- en voortgezet onderwijs aan bod komt en “CT for professionals” of “geavanceerde CT” als het gaat om de meer hogere niveau CT waarbij het ook gaat om het ontwerpen van architecturen, infrastructuren, besturingssystemen etc.

CT is al heel oud maar ook weer niet

Als je de film Hidden Figures gezien hebt, dan weet je dat de eerste “computers” mensen waren. Het woord stamt uit de vroege 17e eeuw en betekende “someone who computes”. De eerste niet menselijke computers werden dan ook “automatic computers” genoemd. De namen van de eerste van zulke computers, zoals de UNIV**AC** en de BIN**AC** verwijzen daar ook naar.
De auteurs geven aan dat ook al in die vroegen dagen van menselijke computers, CT nodig was bij het ontwerp van de benodigde procedures om de benodigde berekeningen uit te voeren. Alleen werd dat toen nog niet zo genoemd. Het eerste gebruik van CT als begrip dat zij hebben kunnen herleiden stamt uit 1980, toch alweer 40 jaar geleden.

Computer Science (CS) en Computational Science

Hoewel CT als concept al eeuwenoud is, werd het pas echt mainstream met de opkomst van het gebruik van de “electronic computer”, de digitale computer zoals wij die nu kennen. Vanaf de jaren 50 van de vorige eeuw ontstond er een industrie waarbij ontwerpers en programmeurs software ontwikkelden voor deze machines. Vanaf de jaren 60 ontstond Computer Science (CS)

als vakgebied waarbinnen CT een belangrijke rol vervulde. In de begintijd ging het nog vooral om het daadwerkelijk aan de praat krijgen van de technologie, maar vanaf de jaren 80 begon het gebruik can CS binnen de andere wetenschapsgebieden en ontstonden vakgebieden als *computational physics* (binnen de natuurkunde), bioinformatics (binnen de biologie), *computational chemistry* (binnen de chemie), *digital humanities *(binnen de geesteswetenschappen) of computational sociology (binnnen de sociologie).
Deze nieuwe gebieden combineerden de oorspronkelijke wetenschapgebieden (bv natuurkunde) met computer science waarbij ze dan bijvoorbeeld numerieke analyses gebruiken om problemen op te lossen in de natuurkunde op basis van bestaande kwantitatieve theorieën. Als een soort tussenweg tussen theoretische en experimentele wetenschap.
Deze nieuwe vakgebieden vergen dat wetenschappers zowel kennis hebben van het primaire domein (bv de natuurkunde) als van CT en CS. In plaats van Computer Science spreken we dan over **Computational Science**.

Dimensies van CT

De auteurs onderscheiden in hun boek zes belangrijke dimensies van CT

1. Methoden (hoofdstuk 2)
2. Machines (hoofdstuk 3)
3. Onderwijs (hoofdstuk 4 + 8)
4. Software ontwikkeling (hoofdstuk 5)
5. Ontwerp (hoofdstuk 6)
6. Computational Science (hoofdstuk 4 + 7)

De hoofdstukken van het boek volgen deze indeling zoals hierboven ook aangegeven gedeeltelijk (de toegevoegde indeling is mijn interpretatie). Ik loop de hoofdstukken even kort langs:

Hoofdstuk 2: Computational Methods

Dit hoofdstuk beschrijf o.a. de noodzaak van decompositie van problemen in kleinere onderdelen Het verwijst naar René Descartes en Gottfried Leibniz als pleitbezorgers van een universele taal die alle misverstanden in communicatie voorkomt en elk conflict door pure logica kan oplossen. Een representatie hiervan is o.a. terug te vinden in Boole's algebra en Boolean logic die sommigen van ons nog wel kennen uit de AND/OR opdrachten die bij vroege zoekmachines nodig waren.

Hoofdstuk 3: Computing Machines

Het hoofdstuk staat eerst stil bij de geschiedenis van de computerhardware om van daaruit te kijken naar de manier waarop software historisch gezien opgebouwd is.
De architectuur van die systemen staat bekend als de “von Neumann architectuur” naar John von Neumann, de notulist van het team dat in 1945 de ENIAC (Electronic Numerical Integrator and Computer) ontwierp. Dit team maakte een onderscheid in drie hoofdcomponenten van een systeem: het geheugen voor opslag van gegevens, een input-output (I/O) systeem en een centrale verwerkingseenheid (een Central Processing Unit of CPU). Deze basisstructuur van computers heeft een zichtbare weerslag op veel definities van CT. Nieuwe architecturen rond kunstmatige intelligentie (neurale netwerken)en quantum-computing passen daarom ook minder goed in veel gehanteerde CT aanpakken.

Hoofdstuk 4: Computer Science

Het onderwerp Computer Science was in de inleiding al even aan bod gekomen, maar krijgt ook in het vervolg een eigen hoofdstuk. Daarbij wordt steeds ook teruggegrepen op de rol die het onderwerp binnen het (universitaire) onderwijs speelt.
Het boek maakt daarbij onderscheid in een aantal tijdvakken:

1. Verschijnselen rond computers (jaren 50 tot 70)
2. Programmeren als kunst en als wetenschap (jaren 70)
3. Computing als middel om te automatiseren (jaren 80)
4. Computing als diepgaande gegevensprocessen (jaren 90 tot nu)

De jaren 50 van de 20e eeuw zagen een gevecht binnen Amerikaanse universiteit voor de inrichting van aparte computerafdelingen en het accepteren van “computing” als aparte academische discipline die zich richt op het bestuderen van verschijnselen (phenomena) die betrekking hebben op computers.
Algoritmisch redeneren ontstond als nieuw aspect van de hierbij benodigde denkwijzen.
De jaren 60 vormden een periode waarin computing volwassener werd. Het deelgebied “netwerk” ontstond in 1967 met de start van het ARPANET project, de voorloper van het internet. Er werden programmeertalen (FORTRAN, LISP, COBOL, ALGOL) ontwikkeld die programmeurs in staat stelden om niet langer in machinetaal programma’s te schrijven.
Eind jaren 70 en begin jaren 80 werd duidelijk dat er heel veel vragen open stonden als het ging om “computing”. Er ontstond een beweging richting het gebruik van computing ten behoeve van automatisering. Daarbij betekende *automatisering* meestal ofwel het uitvoeren van een proces door een machine, ofwel het controleren van processen op een mechanische manier met zo weinig mogelijk menselijke interventie.

De nadruk op dit aspect leidde tot veel pessimisme omdat er heel veel niet te automatiseren was.
Vanaf de jaren 90 en zeker vanaf de 21e eeuw, was computing zó ver de verschillende wetenschapsgebieden binnengedrongen dat duidelijk was dat “automatisering” zeker niet de enige toepassingsmogelijkheid was. Daar ging het namelijk niet (alleen) om het automatiseren van bestaande processen, maar juist meer om het inrichten van nieuwe processen die niet mogelijk zouden zijn geweest zónder computing.

Hoofdstuk 5: Software Engineering

Fouten in computersoftware kunnen tot ernstige problemen leiden. Het boek noemt een voorbeeld uit de ruimteluchtvaart. Maar tegenwoordig zijn door toepassingen van computers in de luchtvaart, gezondheidszorg en transport de gevolgen nóg ernstiger.

Toen de omvang en de complexiteit van de programma’s toenam kwam er veel behoefte aan manieren om de software betrouwbaarder te maken. Het concept van het toepassen van **abstractie** deed begin jaren 70 zijn intrede als manier om de complexiteit behapbaar te maken. Maar ook concepten als **debuggen, recursie, visualisatie** komen nu nog steeds voor als onderdeel van CT.
Het hoofdstuk gaat ook in op de strijd tussen een technicus en een wetenschapper. De technicus wil de detail correct krijgen omdat de software anders niet werkt. De wetenschapper heeft vaak juist interesse in het globale overzicht zodat de samenhang zichtbaar wordt.

Daarnaast in dit hoofdstuk veel aandacht voor de concepten die in de jaren 70 ontstaan zijn als pogingen om het ontwikkelproces van software beter schaalbaar te maken.

Hoofdstuk 6: Designing for Humans

Dit hoofdstuk gaat meer in detail in op wat er komt kijken bij het ontwerpen van software voor mensen.
Een van de begrippen die in het hoofdstuk aan bod komt is **DRUSS** – de ontwerpprincipes voor software:

- **D**ependable,
- **R**eliable,
- **U**sable,
- **S**afe,
- **S**ecure

In de jaren 70 hanteerde de ISO (International Standards Organization) een lijst van 20 meetbare factoren voor het bepalen van de kwaliteit van software: correctness, reliability, integrity, usability, efficiency, maintainability, testability, interoperability, flexibility, reusability, portability, clarity, modifiability, documentation, resilience, understandability, validity, functionality, generality, economy.
De factoren security en safety uit DRUSS stonden nog niet op deze lijst omdat niemand wist hoe die objectief vastgesteld zouden moeten kunnen worden.

In het hoofdstuk wordt de ontwerp gerichte benadering van CT afgezet tegen de software engineering benadering van CT van het hoofdstuk eerder. In sommige gevallen (bv de systemen in een vliegtuig) moet de nadruk op veiligheid / foutvrijheid liggen, in andere gevallen gaat het juist meer om een zo goed mogelijke fit met de behoeften van de gebruikers, ook als daarbij niet aan alle hierboven genoemde ISO-factoren voldaan is.

Hoofdstuk 7: Computational Science

Het hoofdstuk over computational science gaat nog wat verder in op de integratie van computer science in de wetenschappen en het ontstaan van Computational Science. Zie hiervoor ook de eerdere toelichting.

Hoofdstuk 8: Teaching CT for All

Het achtste hoofdstuk van dit boek is weer interessant voor met name het onderwijs. Het start namelijk met een historisch overzicht van de rol van de CT in het onderwijs: eerst uitsluitend op universiteiten, vanaf de jaren 90 ook in andere delen van het onderwijs.
Het hoofdstuk introduceert ook de tegenstelling tussen “computer literacy” en “fluency in ict” maar staat ook stil bij de worstelingen om de initiële claims van pioniers dat computing een “thinking tool for education” is, een vaardigheid die iedereen moet bezitten, via onderzoek ook te onderbouwen.
De transfer van CT als metacognitieve vaardigheid van programmeren naar andere domeinen bleek moeilijk aan te tonen.

Het essay van Jeanette Wing uit 2006 bracht nieuwe belangstelling voor CT met zich mee. Daarbij zaten ook veel enthousiastelingen die niet op de hoogte waren van de voorgeschiedenis van het domein. Dit leidde tot een aantal conflicten tussen groepen nieuwkomers die soms vanaf nul hun eigen raamwerk ontworpen hadden.
Het hoofdstuk geeft een lijst van een zestiental conflicten die er bestaan rond CT. Ter illustratie worden ze hieronder in het Engels weergegeven:

1. Whether CT is limited to thinking about the mechanics of constructing algorithms—or includes thinking about machines, computational science, software engineering, and design.
2. Whether CT is mostly about programming—or also encompasses systems, networks, and architectures; or whether it is not really about any of those.
3. Whether the definition, that CT is the formulation of algorithms to solve problems, is too narrow a view of CT's scope.
4. Whether algorithms are only those that fit the strict definition from the theory of computing – or whether algorithms could also be more loosely defined.
5. Whether algorithms necessarily include an abstract machine in the background.
6. Whether algorithms are primarily directions for controlling machines – or are primarily means of expressing procedures.
7. Whether using computational tools teaches CT.
8. Whether carrying out daily step-by-step procedures is a manifestation of CT.
9. Whether CT is learned from practicing programming – or from welldesigned learning activities that use steps and rules.
10. Whether learning CT in the context of computing transfers to problem-solving skills in other fields.
11. Whether CT is domain dependent – or is a meta-skill valid in all domains.
12. Whether computational processes are found in nature—or whether they are limited to algorithms and machines.
13. Whether information processing by computers differs from information processing done by humans – and whether "information processing agents" can include things such as molecules, DNA, or quarks.
14. Whether students' learning should be assessed from their demonstrating skill at designing computations – or from their knowledge of certain key concepts.
15. Whether satisfaction of customers with the job that software does should be part of the assessment of software success.
16. Whether K–12 CT education has to stick with strict definitions of computing – or could for pragmatic and pedagogical reasons take some liberties.

Het duidelijk dat er genoeg redenen zijn om het heel erg met elkaar oneens te zijn als het gaat om de wijze waarop CT een plek zou moeten krijgen in het onderwijs.

Hoofdstuk 9: Future Computation

In het laatste hoofdstuk komen de auteurs terug op de zaken die er voor zullen zorgen dat computing zich blijft ontwikkelen en afgeleid daarvan zal dat ook voor CT moeten gelden. Eerder werd al genoemd het effect van het belang van "design" (ontwerp), nieuwe

modellen/architecturen naast de bekende “von Neumann architectuur” zoals DNA computing, Quantum-computing, Machine Learning etc.
Belangrijk begrip om hierbij te kennen is **Technology Jumping**, dit is het uitgangspunt dat er op bepaalde momenten een grote sprong in de beschikbare technologie optreedt die er voor zorgt dat Moore’s Law (het verdubbelen van de computerkracht elke twee jaar) veel langer te handhaven blijkt dan verwacht. Voorbeelden zijn bijvoorbeeld het overstappen van ponskaarten naar magneetbanden of van vacuümbuizen naar transistors, etc.

Naschrift: geleerde lessen (van de auteurs)

De auteurs sluiten het boek af met een aantal lessen die ze zelf geleerd hebben tijdens het doen van onderzoek voor het boek.

1. CT voegt iets toe aan de verschillende domeinen, het komt niet in plaats van;
2. CT is een oud, diepgaand onderzocht en divers onderwerp;
3. De snelheid van computers is de belangrijkste veroorzaker van de computerrevolutie;
4. Geavanceerde CT is domeinspecifiek;
5. CT heeft de hulpmiddelen, methoden en de kennistheorie van wetenschap gewijzigd;
6. Als de meeste mensen het over CT hebben, bedoelen ze basale CT;
7. Basale CT + Geavanceerde CT vormen samen een rijke verzameling aan CT;
8. Verandering is onlosmakelijk verbonden aan CT.

Wat heb ik er van geleerd?

Het was een interessant boek om te lezen. Ik kende veel van de geschiedenis over computer science wel, maar had er nog niet zo uitgebreid vanuit een CT-bril naar gekeken.
De link tussen CS en CT en zeker de toevoeging van computational science (al is dat qua woord dan weer onhandig) maakt de keuze voor onderdelen van een aantal van de bekendere indelingen rond CT begrijpelijker en maakt inzichtelijk waarom dingen zo zijn zoals ze zijn. Ook maakt het boek duidelijk dat nieuwe architecturen, zoals neurale netwerken en quantum-computers niet altijd even goed passen bij de huidige CT werkwijze. Het domein zal zich daarbij moeten doorontwikkelen.

De auteurs maken een punt voor wat betreft het opnemen van CT in het domein “computing fluency”, Dat is meer dan alleen ict-geletterdheid, het gaat niet alleen om het kunnen bedienen van apparaten, het vergt dat je vaardig bent in het gebruik van CT daarbinnen. Het Nederlandse “digitale vaardigheden” is dan eigenlijk weer te vaag, omdat de link met “computing” dan niet

duidelijk wordt. Dan is “ict” (zonder hoofdletters, niet als afkorting) daar uiteindelijk toch gewoon beter voor.

Het is daarmee eigenlijk ook een pleidooi om CT niet als een soort generieke, “kan ook zonder een computer” vaardigheid te positioneren. Al lijkt dat op het niveau van “basic CT” zoals we dat in het primair en voortgezet onderwijs tegenkomen minder een probleem.
Maar juist daar lig ook mijn belangrijkste probleem met het boek: ja, er wordt een overzicht gegeven van veelvoorkomende discussiepunten, ja, de auteurs stellen dat ze die allemaal adresseren in hun boek, maar ík kan zo snel niet allemaal terugvinden. En als je me nu, na het lezen van het boek vraagt “hoe moeten we leraren in het primair- en voortgezet onderwijs ondersteunen bij het implementeren van aandacht voor CT in het curriculum?” dan levert het daar geen kant en klare handvatten voor op.