Georges Gielen, Peggy Valcke,
Jan De Bruyne & Victoria Hendrickx

TECHNOLOGIE & RECHT

OWL PRESS
LEGAL

INHOUD

DEEL 1
COMPUTER- EN INTERNET-TECHNOLOGIEËN

DEEL 2
AUTONOME EN INTELLIGENTE SYSTEMEN

DEEL 3
PRIVACY- EN CYBER-BEVEILIGINGSTECHNOLOGIEËN

DEEL 4
BLOCKCHAINTECHNOLOGIE EN SLIMME CONTRACTEN

DEEL V
RECENTE GRENSVERLEGGENDE TECHNOLOGIEËN

VOORWOORD

"Voorspellen is moeilijk, vooral als het om de toekomst gaat."

(Niels Bohr, Deens natuurkundige)

"De beste manier om de toekomst te voorspellen is deze te creëren."

(Peter F. Drucker, management consultant, Verenigde Staten)

"We zijn geneigd het effect van een technologie op de korte termijn te overschatten en het effect op lange termijn te onderschatten."

(Roy C. Amara, Institute for the Future, Verenigde Staten)

De technologische ontwikkelingen in onze samenleving volgen elkaar in een razendsnel tempo op: nano-elektronica en supersnelle computers, het internet en de cloud, artificiële intelligentie en slimme systemen, deepfakes en grote taalmodellen, blockchain, cryptomunten en slimme contracten, robotica en autonome systemen, virtuele werelden en metaversums, genetische manipulatie, noem maar op. Elk van ons krijgt onvermijdelijk in zijn of haar professionele én persoonlijke leven met deze evoluties te maken. In dit boek richten we ons vooral op technologische evoluties op het vlak van de informatie- en communicatietechnologie (ICT). De huidige digitale transformatie heeft nu al een ingrijpende impact op onze manier van leven, wonen en werken, en dit zal de komende jaren alleen maar toenemen. Technologische ontwikkelingen brengen nieuwe mogelijkheden en opportuniteiten met zich mee die we moeten grijpen, maar ze gaan tegelijkertijd gepaard met heel wat sociaal-maatschappelijke, ethische en juridische uitdagingen. Ze maken ons als individuele burgers kwetsbaarder, onder meer door de toenemende druk op onze privacy en veiligheid.

Niemand kan expert zijn in al deze domeinen. Toch is het belangrijk dat ook niet-experten de grote lijnen op het vlak van technologie begrijpen en op basis daarvan in staat zijn om de impact, de mogelijkheden en de ethisch-juridische dilemma's te begrijpen. Ten eerste is dit belangrijk om inzicht te krijgen in de huidige ingrijpende veranderingen in het eigen vakgebied, en tegelijk ook in de opportuniteiten die deze ontwikkelingen met zich meebrengen. Geen enkel beroepsgebied is immers immuun voor de evoluties in ICT en de digitalisering. Om het met wat beeldspraak uit te drukken, maken we de vergelijking met een grote berg sneeuw. De technologische ontwikkelingen 'knabbelen' langzaam aan de buitenkant van de berg, en mogelijks voelt men dat niet in het centrum van de berg, maar smelten zal ie! Bijgevolg volstaat

het niet langer om specialist te zijn in een bepaald domein zonder te beschikken over enige digitaal-technologische geletterdheid. Dit boek beoogt dat hiaat op te vullen door een bredere ICT-technologische inkijk aan te bieden voor wie een niet-technologische opleiding volgt of gevolgd heeft. Ten tweede verschaft het boek de lezer het nodige inzicht om vanuit het eigen vakdomein in gesprek te kunnen gaan met de technologieontwikkelaars, zodat men voor het eigen domein voldoende nuttige en efficiënte oplossingen kan bekomen. In de gezondheidszorg noemen we dergelijke oplossingen *healthtech*, in het juridische domein *legaltech*, in het financiële domein *fintech*, enzoverder. Het boek is een hulpmiddel om de brug te helpen slaan tussen de technologen die de ICT-oplossingen dienen te ontwikkelen en de niet-technologen die uiteindelijk die oplossingen dienen te gebruiken. Ten derde wenst het boek de lezer voldoende te wapenen om vanuit de eigen vakexpertise te kunnen deelnemen aan interdisciplinaire discussies en (beleids)keuzes rond nieuwe technologieën. Ook deze aspecten komen in dit boek aan bod. Dat op die manier meer personen vanuit diverse disciplines aan het debat kunnen deelnemen, kan de besluitvorming en wetgeving over nieuwe technologieën – waarbij belangrijke afwegingen moeten worden gemaakt tussen maatschappelijke en economische voordelen enerzijds en risico's en ethische bezorgdheden anderzijds – alleen maar ten goede komen.

Het boek is ontstaan uit de cursus voor het vak Technologie en Recht dat sinds 2018-2019 als plichtvak gedoceerd wordt aan alle studenten in de eerste master aan de faculteit Rechten en Criminologische Wetenschappen aan KU Leuven. De blik onder de motorkap die het vak geeft, moet de (toekomstige) jurist in staat stellen kritisch te reflecteren over de technologische ontwikkelingen en hun maatschappelijke impact – met inbegrip van de invloed van die ontwikkelingen op het recht en de beroepspraktijk zelf – én in dialoog te treden met technische experten uit de behandelde domeinen. De impuls voor het vak kwam van de toenmalige decaan Bernard Tilleman en de onderwijsprogrammadirecteur Vincent Sagaert die vonden dat de rechtenstudent niet langer kon afstuderen zonder een basisvorming in digitale geletterdheid. De jurist van de toekomst dient immers een T-gevormde (*T-shaped*) jurist te zijn met naast een brede disciplinaire kennis ook verbredende kennis in relevante aanpalende disciplines en vaardigheden, waaronder digitale kennis en vaardigheden. De beroepspraktijk van de jurist is onder invloed van de digitale technologieën immers grondig aan het veranderen. Bijgevolg moeten juristen kunnen omgaan met deze nieuwe technologieën en ethische dilemma's, en moeten zij in staat zijn om te functioneren in interdisciplinaire teams, waarvan in de toekomst ongetwijfeld ook *legal engineers* deel zullen uitmaken. Na wat initiële discussies waaraan ook Phil Dutré, vicedecaan onderwijs van de faculteit

Ingenieurswetenschappen, deelnam, werden Georges Gielen en Peggy Valcke, twee van de auteurs van het boek, met de zonet vermelde doelstellingen als docenten van het vak aangesteld. Zij hebben het vak concreet inhoud gegeven, de onderwerpen en gastsprekers geselecteerd, en vervolgens de bijhorende cursustekst in een eerste iteratie uitgewerkt. Ondertussen zijn er al zes jaargangen gepasseerd en heeft de tekst een hele evolutie doorlopen. Recent heeft collega Jan De Bruyne het vaste docententeam vervoegd, en op dit moment fungeert Victoria Hendrickx als medewerkster voor het vak. Het is dit gemengd technologisch-juridische team dat uiteindelijk met veel zwoegen (een boek is toch wel iets complexer dan een cursustekst en je krijgt er aan de universiteit geen vrijstelling in tijd voor!) de boekversie voltooid heeft.

Vanaf de start van het vak is er geopteerd om er geen vak ICT-*recht* van te maken (dat vak bestaat namelijk al en is meer specialistisch gericht), en evenmin een vak programmeren voor juristen (het is wellicht niet nodig dat elke jurist zelf kan programmeren), maar wel om een evenwichtige balans aan te bieden tussen technologie en recht. Vandaar ook de titel van het vak Technologie en Recht. Het daaruit resulterende boek neemt dit evenwicht en deze titel over: het bespreekt enerzijds de technologische ICT-bouwstenen en haalt anderzijds de bijhorende juridisch-ethische overwegingen en regelgeving aan. Daarmee probeert het de brug te leggen tussen juridisch geschoolde en ICT-technisch geschoolde experts. De grote moeilijkheid was om een keuze te maken in de onderwerpen die wel dan niet behandeld worden in het vak en in het boek, en de mate van diepgang waarmee elk onderwerp behandeld wordt. Kiezen is verliezen, zegt men. De uiteindelijke keuze was gebaseerd op relevantie en impact. De brede onderwerpen die uiteindelijk aan bod komen (en die overeenkomen met de verschillende delen in het boek) zijn computer- en internettechnologieën, artificiële intelligentie en autonome systemen, cyberbeveiliging en online privacy, blockchain en slimme contractafsluiting, en tot slot biotechnologie en recente digitale evoluties zoals virtuele werelden. De cursustekst is samengebracht uit en gebaseerd op vele bronnen, waar we uitvoerig in de tekst naar verwijzen (ofwel in voetnoot ofwel in de referenties achteraan). Vanzelfsprekend staan de behandelde domeinen niet stil, noch aan technologische, noch aan juridische kant. Daarom anticiperen we dat het boek om de twee jaar bijgewerkt dient te worden.

Een onverwachte uitdaging was de taal. Technologie en ICT in het bijzonder zijn domeinen bij uitstek waarin vele Engelstalige termen gehanteerd worden. In de mate van het mogelijke hebben we de tekst in het Nederlands geschreven, met vermelding van de overeenkomstige Engelse bewoordingen (steeds in *italic* aangeduid). Her en der

zijn de Engelstalige termen zodanig ingeburgerd dat er geen Nederlandse vertaling meer nodig of gebruikelijk is. Een gelijkaardig probleem vormden de figuren. Soms konden de rechten op een gewenste figuur niet verkregen worden door de uitgever, en diende een alternatief gezocht dat niet altijd dezelfde lading dekt. Ondanks het nazicht door vele personen zullen typo's ongetwijfeld nog opduiken in deze eerste editie. Deze mogen met plezier aan de auteurs gemeld worden (zie het e-mailadres onderaan dit voorwoord). Ook inhoudelijke suggesties ter verbetering of aanvulling zijn steeds welkom.

Tot slot wensen we nog diverse personen te bedanken, in het bijzonder de sprekers (buiten de auteurs) van de lessen in de zes voorbije academiejaren (in alfabetische volgorde): Thomas Aertgeerts, Herman Bruyninckx, Nadine Buys, Erik De Herdt, Claudia Diaz, Phil Dutré, Carolien Michielsen, Sien Moens, Yves Moreau, Frank Piessens, Bart Preneel, Maarten Truyens, Tinne Tuytelaars, Toon Vanagt, Geert Van Calster, Tim Van de Cruys, Gerrit Vandendriessche, Geneviève Vanderstichele, Luc Van Gool, Erik Valgaeren, Mathy Vanhoef, Dries Wijnen en Kim Wuyts. We danken hen voor hun bijdrage tot de lessen en de opbouw van het materiaal. Ze hebben ook heel wat referentieliteratuur gesuggereerd. Over de zes jaargangen heen hebben we individuele bijdragen her en der moeten aanpassen en sommige onderdelen zijn toegevoegd of verwijderd op basis van de ervaringen en de evoluties van het domein. We danken ook de collega's uit binnen- en buitenland van wie we tekstmateriaal als bron hebben mogen putten. Tevens danken we Lies Van Eycken en Dirk Van Havermaet van de dienst permanente vorming van de rechtsfaculteit. Dankzij hen maakten we al video-opnames van alle lessen van dit vak lang voor Covid-19 dit tot standaard verhief. Bovendien stelden we via hen ook de interesse in het materiaal van dit boek vast van personen die al in de praktijk staan en van een breder publiek in het algemeen. Dit was een sterke motivatie om het materiaal uiteindelijk om te vormen tot een echt boek. Van harte dank ook aan de verschillende generaties van studenten en medewerkers die over de jaren heen hebben meegewerkt aan de opbouw, revisie en correctie van de cursustekst, in het bijzonder (in alfabetisch volgorde): Amber Boes, Caroline Calomme, Dieter Decraene, Hannes Dewaele (die de motor achter de initiële cursuswiki vormde) en Rune Vanleeuw. We wensen ook onze waardering uit te drukken voor het logistieke team van Owl Press Legal onder leiding van Paulien Vandenberghe en Nancy Debraekeleer dat instond voor het corrigeren van het manuscript en de finale vormgeving van het boek. Op die manier is het eindresultaat – zo hopen wij – een aantrekkelijk boek geworden dat zowel studenten als een breder publiek aanspreekt

én dat aanzet tot verdere exploratie voor zij die gebeten zijn door de wondere wereld van technologie en recht.

Tot slot betrekken we graag onze familie en collega's in onze woorden van erkenning, want als het puntje bij paaltje komt, hebben wij toch finaal bij hen tijd afgeknabbeld om de verschillende stappen in de totstandkoming van dit boek tijdig te kunnen voltooien.

Veel leesplezier! En stuur gerust opmerkingen en suggesties naar het volgende e-mailadres: technologie-en-recht@kuleuven.be.

De auteurs Georges Gielen, Peggy Valcke, Jan De Bruyne en Victoria Hendrickx,
1 januari 2025.

OVER DE AUTEURS

Prof. dr. ir. Georges Gielen is burgerlijk elektrotechnisch-werktuigkundig ingenieur, gewoon hoogleraar aan KU Leuven in de onderzoeksafdeling MICAS (Microelectronics And Sensors) van het departement Elektrotechniek (ESAT). Hij is voormalig vicerector voor de Groep Wetenschap & Technologie. Zijn onderzoeksspecialisatie is in micro- en nano-elektronica en CAD-softwaretechnieken voor het ontwerp van elektronische geïntegreerde schakelingen, met klemtoon op analoge en gemengd analoog-digitale schakelingen. Hij is actief in tal van onderzoeksprojecten rond elektronica en slimme/autonome systemen. Verder is hij is verbonden met imec, het Vlaamse onderzoekscentrum voor nanotechnologie. Hij is Fellow van IEEE, lid van de Koninklijke Vlaamse Academie van België (KVAB) in de klasse van de Technische Wetenschappen, lid van de Academia Europaea en ontving talrijke onderscheidingen voor zijn onderzoek. Hij is co-editor van het boek *AI and contracting*.

Prof. dr. Peggy Valcke is deeltijds gewoon hoogleraar aan de Faculteit Rechtsgeleerdheid en Criminologische Wetenschappen van KU Leuven. Ze is codirecteur van de onderzoeksgroep imec-KU Leuven-CiTiP (Centre for IT & IP Law). Sinds januari 2024 is ze Raadslid van het Belgisch Instituut voor Postdiensten en Telecommunicatie (BIPT). Haar expertise ligt op diverse domeinen van het ICT-recht, met een focus op de reglementering van mediaplatformen, artificiële intelligentie, de bescherming van persoonsgegevens, cyberbeveiliging en het beheer van data. Tussen 2008 en 2022 was ze lid van de Vlaamse Regulator voor de Media en assessor bij de Belgische Mededingingsautoriteit. Ze vertegenwoordigde België op het niveau van de Raad van Europa in het Comité voor AI bij de uitwerking van een juridisch kader voor artificiële intelligentie, en fungeerde van 2019 tot 2021 als vice-voorzitter van het Ad Hoc Comité voor AI (CAHAI).

Prof. dr. Jan De Bruyne is docent IT-recht aan de Faculteit Rechtsgeleerdheid en Criminologische Wetenschappen van KU Leuven. Hij is het hoofd van de onderzoeksgroep imec-KU Leuven-CiTiP (Centre for IT & IP Law) en eveneens codirecteur van het Vlaams Kenniscentrum Data & Maatschappij. Hij is de *Principal Investigator* (PI) van talrijke nationale en Europese projecten, waarin hij zich toelegt op de juridische en ethische uitdagingen veroorzaakt door technologische evoluties. Hij heeft een bijzondere interesse in vraagstukken over de buitencontractuele aansprakelijkheid voor schade veroorzaakt door autonome systemen. Dat heeft geleid tot talrijke publicaties in (inter)nationale tijdschriften en boeken, zoals *Autonome motorvoertuigen: een multidisciplinair onderzoek naar de maatschappelijke impact, Artificiële intelligentie en Maatschappij en Artificial intelligence and the law.*

Victoria Hendrickx is doctoraatsonderzoeker van KU Leuven verbonden aan de onderzoeksgroep imec-KU Leuven-CiTiP (Centre for IT & IP Law). Haar onderzoek richt zich op de invloed van digitale technologieën op de rechtspraak, en in het bijzonder op de rechterlijke motiveringsplicht. Sinds 2022 assisteert ze bij de KU Leuven Summer School on the Law, Ethics and Policy of AI en is ze co-editor van de bijbehorende blog *Law, Ethics & Policy of AI*.

DEEL 1

COMPUTER- EN INTERNET-TECHNOLOGIEËN

Korte samenvatting

Dit eerste deel van het boek bespreekt de evolutie, de werking en het gebruik van computersystemen en -netwerken, evenals de basisbegrippen van softwaresystemen en -applicaties. Samen vormen ze de motor achter de digitale transformatie die momenteel volop in onze samenleving aan de gang is en die een impact heeft en verder zal hebben op al wat we doen, inclusief onze rechtspraktijk. We gaan ook dieper in op de manier waarop het internet als netwerk *par excellence* functioneert (de structuur van het internet) en beheerd wordt (de *governance* van het internet). Ook komen de verschillende types van *cloudcomputing* aan bod, waarbij via het netwerk – veelal het internet – hardware, software of gegevens op aanvraag ter beschikking worden gesteld. Tot slot komen ook de onderliggende technieken, voordelen en risico's van het Internet der Dingen (*Internet of Things*) aan bod.

HOOFDSTUK 1 COMPUTERSYSTEMEN, ALGORITMES EN SOFTWARE

1. INLEIDING

Computers en de software die ze uitvoeren vormen de motor van de **digitale transformatie** die momenteel volop in onze samenleving aan de gang is en die een impact heeft op al wat we doen. In dit hoofdstuk gaan we dieper in op de evolutie, de werking en het gebruik van computersystemen en -netwerken, evenals op de basisbegrippen van softwaresystemen en -applicaties. Naast de huidige werking en mogelijkheden van computers wordt in dit hoofdstuk ook een blik geworpen op toekomstige technologieën voor computersystemen, waarbij ontwikkelingen zoals neuromorfe en kwantumcomputers aan bod komen.[1] Ook wordt het begrip en de werking van algoritmes uitgelegd[2] en worden het proces van softwareontwikkeling en -verbetering, het verschil tussen broncode en uitvoeringscode en het concept van openbronsoftware beschreven.[3]

2. COMPUTERSYSTEMEN

Onze samenleving wordt momenteel gekenmerkt door een digitale revolutie, ook wel de digitale transformatie genoemd.[4] Die digitale revolutie wordt gedragen door computers, een concept waar we dieper op ingaan.[5] We bespreken hoe de huidige computers zijn

1 Zie sectie 2 in dit hoofdstuk.
2 Zie sectie 3 in dit hoofdstuk.
3 Zie sectie 4 in dit hoofdstuk.
4 Zie sectie 2.1 in dit hoofdstuk.
5 Zie sectie 2.2 in dit hoofdstuk.

opgebouwd met micro/nano-elektronica,[6] en vermelden ook enkele nieuwe technologieën die momenteel in onderzoek zijn om nog krachtigere computers te kunnen realiseren.[7]

2.1. DE DIGITALE REVOLUTIE

De **digitale revolutie** is een kernpunt in het huidige beleid van de Europese Unie.[8] Deze digitalisering heeft nu al een ingrijpende impact op onze manier van leven, wonen en werken, en zal dat in de komende jaren nog veel meer hebben. Dat gaat van de digitale verwerking van data en documenten (bv. belastingaangiftes, notarisaktes, dokters-attesten, betalingsbewijsjes, enz.) over de digitalisatie van de processen in bedrijven en organisaties (bv. e-government, elektronisch betalen, online bestellen, virtueel les volgen, automatisatie in magazijnen en bedrijfshallen, papierloze kantoren, toegang tot vliegtuigen, musea of festivalweides zonder fysieke tickets, enz.) tot de volledige digitalisatie van onze samenleving zelf (d.w.z. volledig leven in een digitale cultuur, waarbij ook contacten en communicatie hoofdzakelijk via de cyberwereld verlopen).[9]

Deze digitale transformatie impliceert dat veel processen die vroeger fysiek werden uitgevoerd, voortaan digitaal worden uitgevoerd. Dat heeft niet alleen sociaaleconomisch een grote impact, maar het kan de bestaande markt grondig omgooien (bv. resulteren in enkele andere economische spelers die voortaan de markt gaan domineren) en het vergt ook mentaal-cultureel een grote omslag van mensen. Maar de voordelen (economisch efficiënter, data en informatie overal en altijd bereikbaar en makkelijk deelbaar met meerdere personen/entiteiten, geen papier en stockage noch bijhorend transport, enz.) worden groter ingeschat dan de nadelen, en zeker jongeren leven al volop in deze digitaliteit.

Onderliggend op technologisch vlak wordt die digitale transformatie in de samenleving mogelijk gemaakt door de enorme vooruitgang in de informatie- en communicatietechnologie (gezamenlijk ICT genoemd), zowel inzake de ICT-hardware (computers en hun netwerken) als de ICT-software. Deze evoluties worden in de rest van dit eerste deel van het boek verder besproken.

6 Zie sectie 2.3 in dit hoofdstuk.
7 Zie sectie 2.4 in dit hoofdstuk.
8 Zie daarvoor de vele documenten van de Europese Commissie inzake de Europese digitale strategie en de Europese digitale transformatie.
9 Die volledig gedigitaliseerde vorm van samenleving werd door Nicolas Negroponte de digitaliteit genoemd, naar analogie met de begrippen moderniteit en postmoderniteit (Negroponte, 1995).

2.2. COMPUTERS

De digitale revolutie wordt gedragen door **computers** (hardware én software) die via netwerken (wereldwijd) verbonden zijn. Computers en (micro)controllers[10] komen natuurlijk op veel wijzen voor. Veelal zijn ze niet eens zichtbaar voor de gebruiker (bv. in huishoudapparaten of in je auto, of de computers van Google die je als gebruiker aan het werken zet als je op je smartphone of laptop een zoekopdracht in je Google Chrome internetbrowser ingeeft).

In de loop van zijn geschiedenis heeft de mens altijd belangrijke technologische uitvindingen gedaan en apparaten en hulpmiddelen ontwikkeld om zijn eigen beperkte mogelijkheden uit te breiden. Van landbouw (voeding) en het wiel (mobiliteit) over het schrift en het gedrukte boek (communicatie) en de stoommachine en de elektrificatie (arbeidskracht) tot moderne varianten zoals de telefoon, radio en televisie, de ontploffingsmotor, de auto en het vliegtuig, de computer en het internet (informatie en data), om maar enkele belangrijke realisaties te noemen. Maar de computer is fundamenteel verschillend van andere uitvindingen van technologische apparaten, omdat de computer via software programmeerbaar is voor meerdere toepassingen. Daartegenover kan bijvoorbeeld een broodrooster alleen worden gebruikt om brood te roosteren en een fiets om te fietsen. Vandaar onderstaande omschrijving.

Een **computer** is een hardwaremachine om berekeningen uit te voeren, die met software programmeerbaar is om op dezelfde machine meerdere programma's te kunnen uitvoeren. Het is een apparaat waarmee gegevens (data) volgens formele procedures (algoritmes) verwerkt/bewerkt kunnen worden om tot het gewenste resultaat te komen.

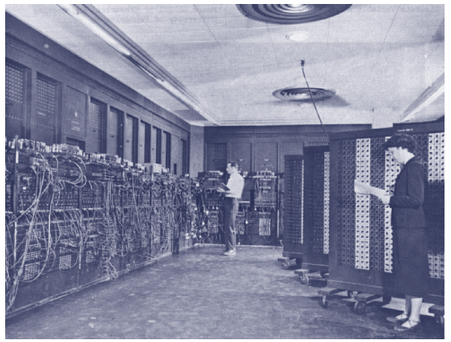
In de volgende secties geven we een kort overzicht van de evolutie van computers[11] en bespreken we de standaard von Neumann-computerarchitectuur.[12]

10 (Micro)controllers zijn kleinere computers ingesteld voor specifieke taken, zoals de aansturing van een wasmachine of een auto.
11 Zie sectie 2.2.1 in dit hoofdstuk.
12 Zie sectie 2.2.2 in dit hoofdstuk.

2.2.1. De evolutie van computers

De eerste programmeerbare rekenmachines of computers werden al in de 19e eeuw ontwikkeld, zij het destijds met behulp van mechanische componenten (bv. de Babbage-machine). Pas in de 20e eeuw werden elektronische componenten gebruikt: eerst elektronenbuizen, vervolgens halfgeleidercomponenten (of transistoren). Firma's zoals IBM (opgericht in 1911) waren toen toonaangevend. De Tweede Wereldoorlog en vervolgens de Koude Oorlog waren grote drijfveren om almaar krachtigere computers te bouwen (bv. om de code te kraken waarmee de 'vijand' boodschappen versleutelde, of om de ballistische trajecten van raketten te berekenen, enz.).

De eerste elektronische computers met elektronenbuizen zoals de Colossus (Verenigd Koninkrijk, 1943) en de ENIAC (Verenigde Staten van Amerika, 1946 – zie figuur 1.1) waren niet alleen groot, maar verbruikten ook veel vermogen en waren weinig bedrijfszeker. Deze nadelen verbeterden sterk met de komst van de halfgeleidertransistor die in 1947 werd uitgevonden.[13] Zeker met de ontwikkeling en de geïntegreerde schakeling[14] (ook wel chip of IC genoemd) en de CMOS-halfgeleidertechnologie werd het mogelijk om snellere en betrouwbaardere computers te maken die minder vermogen verbruiken.



Figuur 1.1: In de beginjaren vulden computers een volledige zaal. Foto van de ENIAC, de eerste computer van de V.S.A. (1946).

13 John Bardeen, Walter Brattain en William Shockley (van AT&T Bell Labs) kregen in 1956 de Nobelprijs voor Natuurkunde voor hun ontwikkeling van de eerste transistor. Die transistor was van het bipolaire type. Pas later werd de MOS-transistor ontwikkeld.

14 Jack Kilby (van Texas Instruments) kreeg er in 2000 de Nobelprijs voor Natuurkunde voor. Min of meer in parallel ontwikkelde Robert Noyce (van Fairchild Semiconductor) hetzelfde concept, maar hij overleed voor de Nobelprijs werd toegekend.

Figuur 1.2: Foto's van verschillende types van computers: (links) de Honeywell-Bull DPS 7-mainframe computer uit 1981, (midden) de Apple II personal computer, en (rechts) een typische laptop uit 2004. die op de schoot kan worden gebruikt.

Ten gevolge van de almaar verdergaande schaalverkleining van de transistoren gebruikt in de CMOS-**halfgeleidertechnologie** werden de computers over de jaren heen almaar kleiner én tegelijk krachtiger in rekenkracht (zie verderop de wet van Moore). Dat leidde tot de ontwikkeling van de persoonlijke computer (bv. de Apple II werd in 1977 gelanceerd en de eerste IBM PC in 1981) en later van de laptop, het netbook en het tablet. Bedrijven zoals Intel (opgericht in 1968) en Motorola (opgericht in 1928, maar sinds 1974 actief in computerchips) en later Apple (opgericht in 1976) waren daarbij de koplopers. Zie enkele voorbeelden van verschillende modellen van computers in figuur 1.2 – dit schetst ook mooi de evolutie over de jaren heen. Tegelijk ontstonden de eerste bedrijven die software aanboden als product. Een voorbeeld is Microsoft (opgericht in 1975), dat een besturingssysteem (initieel het Microsoft Disk Operating System (MS DOS) en later het Windows besturingssysteem) en bijbehorende applicatiesoftware (bv. MS Word of Microsoft Office als kantoorsoftwarepakket) aanbood en nog altijd verkoopt. Andere softwarebedrijven zoals Oracle (opgericht in 1977) en Adobe (opgericht in 1982) volgden met nog andere softwaretoepassingen. Naast commerciële producten ontstonden er parallel ook openbronversies van besturingssystemen (zoals het in 1969 aan Bell Labs ontworpen UNIX en later het daaraan gerelateerde Linux) en van softwareapplicaties. Naarmate de computers krachtiger werden en via het internet genetwerkt werden, werden ook meer en meer softwarefirma's met nieuwe diensten en softwaretoepassingen opgericht: Amazon (in 1994), Netflix (in 1997), Google (in 1998), Facebook (in 2004), Youtube (in 2005), Spotify (in 2012), OpenAI (in 2015), enzovoort. Het overwicht in termen van beurswaardering is sindsdien helemaal overgeslagen naar de aanbieders van softwarediensten via het internet, eerder dan naar de computerfabrikanten.[15] De diverse Chips Act-initiatieven

15 Bestaande bedrijven zoals IBM hebben zich om dezelfde reden ook geheroriënteerd in de richting van het aanbieden van diensten, eerder dan het verkopen van hardware zoals persoonlijke computers.

in veel landen of regio's (waaronder de EU) luiden ondertussen wel de alarmbel dat we geostrategisch niet al te afhankelijk mogen worden van het buitenland inzake elektronicaontwerp en -productie.[16]

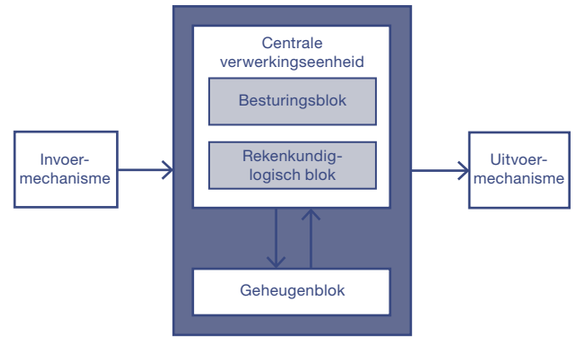
2.2.2. De von Neumann-computerarchitectuur

De huidige computers gebruiken typisch de von Neumann-architectuur die gebaseerd is op wat in 1945 door de wis- en natuurkundige John von Neumann en collega's werd beschreven in het document '*First draft of a report on the EDVAC*' (von Neuman, 1945). Daarin wordt een architectuur voor een elektronische digitale computer beschreven. Die computer werd nadien met succes gerealiseerd en de architectuur wordt nu nog altijd gebruikt in de meeste computers voor algemeen gebruik.

De **von Neumann-computer** bestaat uit (zie figuur 1.3):

1. een centrale verwerkingseenheid (*Central Processing Unit* of CPU genoemd), bestaande uit:
   - een rekenkundig-logische eenheid *(Arithmetic/Logic Unit* of ALU genoemd) met processorregisters (tijdelijke lokale geheugens) voor de berekeningen;
   - een besturingseenheid (*Control Unit*) met instructieregister en programmateller voor de aansturing;
2. een computergeheugen (*RAM Memory Unit*) om zowel de data als de instructies op te slaan;
3. invoer- en uitvoermechanismen (*Input en Output Devices*) en externe massaopslag van gegevens en bestanden.

[16] Zie daarvoor: https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/european-chips-act_nl.



Figuur 1.3: Basisschema van de von Neumann-architectuur voor een computer.

Een dergelijke von Neumann-computer werkt als volgt. De computer wordt geprogrammeerd met een softwareprogramma (een reeks van instructies die achtereenvolgens worden uitgevoerd op de ingevoerde gegevens). Dat programma wordt door de programmeur(s) opgesteld en vervolgens ingelezen en opgeslagen in het geheugen van de computer. De CPU haalt een voor een de programma-instructies uit het geheugen en voert ze uit op de data die ingevoerd worden (die bv. ingetypt worden of ingelezen uit een bestand) en die eveneens in het geheugen van de computer worden opgeslagen. De hoofdstappen van elke instructiecyclus[17] zijn dus het opladen van de betrokken instructie vanuit het geheugen in het instructieregister, het decoderen van de instructie in concrete controlesignalen voor de blokken die men in de ALU nodig heeft en het effectief uitvoeren van de instructie op die blokken, veelal gevolgd door het wegschrijven van de resultaten in het geheugen. Aangezien de meeste computers digitaal werken, moeten ingevoerde gegevens eerst worden gedigitaliseerd (bv. foto's, video en muziek maar ook sensor- of communicatiesignalen moeten eerst worden gedigitaliseerd, d.w.z. omgezet in een reeks van bits (enen '1' en nullen '0') waar de digitale computer mee kan werken). Resultaten van (tussen)berekeningen in het programma worden opnieuw naar het computergeheugen weggeschreven. Het eindresultaat van de berekeningen wordt uiteindelijk uitgevoerd en bijvoorbeeld op het scherm getoond of in een bestand weggeschreven. De processoren of CPU's die als centraal bouwblok in computers gebruikt worden en die ontworpen en gefabriceerd worden door firma's zoals Intel (bv. de Core i9 chip) en AMD (bv. de Ryzen 9 chip), worden met de jaren almaar krachtiger in rekencapaciteit (zie de wet van Moore in de volgende sectie). Ook de opslagcapaciteit van het geheugen in computers neemt

[17] In het Engels spreekt men van de *load-decode-execute* instructiecyclus, vaak gevolgd door een *store* stap.

met de jaren almaar toe. Beide zijn een gevolg van de voortdurende schaalverkleining van de halfgeleidertechnologie.

Merk op dat een moderne computerchip eigenlijk meerdere processorkernen naast elkaar bevat die alle met hetzelfde geheugen verbonden zijn. Dergelijke multikernprocessoren[18] laten toe om softwareprogramma's nog sneller, of om verschillende taken in parallel, te kunnen uitvoeren. Ook bevat een computer naast de algemene CPU voor numerieke en logische bewerkingen meestal ook bijkomende gespecialiseerde processoren voor specifieke taken, zoals een DSP (*Digital Signal Processor*) voor digitale signaalverwerkingstaken, een GPU (*Graphical Processing Unit*) voor grafische bewerkingen (bv. om het scherm aan te sturen) of een neurale eenheid voor toepassingen van artificiële intelligentie. Zie voorbeelden hiervan in het hoofdstuk over tekst- en beeldanalyse en -generatie.[19] Door de vorderingen in de halfgeleidertechnologie[20] worden die verschillende blokken veelal op één chip geïntegreerd, zoals de M2 Max chip van Intel die zowel een multikern CPU, een multikern GPU als een multikern neurale eenheid omvat naast een grote hoeveelheid lokaal geheugen.

Bij de von Neumann-architectuur kan de ophaling van de volgende instructie die moet worden uitgevoerd en de uitvoering van een bewerking op de data niet op hetzelfde moment plaatsvinden, aangezien ze een gemeenschappelijke verbinding of bus naar het computergeheugen delen (zie de dubbele pijl in het schema van figuur 1.3). De eenvoud van de architectuur is het grote succes van de von Neumann-computer, maar tegelijk kan die gezamenlijke toegang van de CPU naar het geheugen voor zowel instructies als data in de praktijk soms een flessenhals zijn die de snelheid beperkt waarmee de computer zijn programma kan uitvoeren. Vandaar dat momenteel ook andere computerarchitecturen bestudeerd en onderzocht worden. Verderop[21] zullen er twee worden besproken: neuromorfe en kwantumcomputers.

2.3. ELEKTRONICA EN DE WET VAN MOORE

De huidige computers zijn opgebouwd met **micro/nano-elektronica**. Het basisbouwblok waaruit elektronica is opgebouwd, is de transistor. Fysiek wordt die transistor

[18] Dit wordt ook wel multicoreprocessoren genoemd.
[19] Zie hoofdstuk 4 in dit boek.
[20] Zie sectie 2.3 in dit hoofdstuk.
[21] Zie sectie 2.4 in dit hoofdstuk.

gerealiseerd in een halfgeleidertechnologie. Wegens haar lage vermogenverbruik wordt voor computerchips momenteel de CMOS-technologie gebruikt, met silicium als basishalfgeleidermateriaal. Een transistor kan dan eigenlijk worden beschouwd als een elektronische versie van een digitaal programmeerbare schakelaar: die kan worden aangeschakeld (geleiden, dus een logische bit '1') of afgeschakeld (niet geleiden, dus een logische bit '0') om zo digitale berekeningen uit te voeren. Alle te verwerken invoergegevens (getallen, tekst, foto's, geluid, enz.) moeten dus eerst worden voorgesteld als reeksen van bits ('*binary digits*', d.w.z. nullen en enen) om in computers verwerkt te kunnen worden. Door dan transistoren op de gepaste manier te combineren kunnen zo in principe alle rekenkundige en logische bewerkingen op die bits worden uitgevoerd. Het mooie aan computers is dat ze programmeerbaar zijn: welke bewerkingen op elk moment in de tijd precies moeten worden uitgevoerd, wordt bepaald door de instructies van het softwareprogramma dat wordt uitgevoerd. Om een volledige computer met rekenprocessoren en geheugens te bouwen, zijn dus veel transistoren nodig die met elkaar verbonden moeten kunnen worden. Om de kostprijs daarvan te drukken, worden die transistoren zoveel mogelijk gezamenlijk in grote aantallen op één geïntegreerde schakeling (een '*integrated circuit*' (IC) of 'chip' genoemd) gefabriceerd, die dan apart wordt verpakt om snelle montage in een computersysteem mogelijk te maken. Een computer of bij uitbreiding elk elektronisch apparaat kan dan een of meer van dergelijke verpakte chips bevatten, typisch gemonteerd op een gedrukt bordje ('*printed circuit board*' of PCB). De chips zelf worden op economisch zeer efficiënte manier gefabriceerd in speciale hooggesofisticeerde chipfabrieken. Aan KU Leuven is de onderzoeksgroep MICAS[22] op wereldniveau toonaangevend in het onderzoek naar innovatieve chips, terwijl het Vlaamse onderzoekscentrum imec baanbrekend is op het vlak van halfgeleidertechnologie.

Gordon Moore, medeoprichter van de firma Intel, voorspelde in 1965 dat het aantal transistoren op een CMOS-chip elke anderhalf jaar zou verdubbelen en dit dankzij de continue verkleining (of schalering) van de transistoren (Moore, 1965). Later is dat tempo bijgesteld naar een verdubbeling elke twee jaar. Men noemt dat de '**wet van Moore**' (hoewel het eigenlijk eerder een voorspelling dan een wetmatigheid is – de industrie heeft het wel gedurende vele decennia tot op de dag van vandaag als een zelfverklaarde leidraad en dus als een soort wetmatigheid gebruikt). Door die voortdurende schaalverkleining van de transistoren worden de computers almaar sneller, waardoor ze meer en meer rekenkracht verkrijgen, d.w.z. almaar meer bewerkingen per seconde kunnen uitvoeren, voor dezelfde prijs. De kleinste CMOS-transistoren

[22] Zie www.esat.kuleuven/micas en www.imec.be.

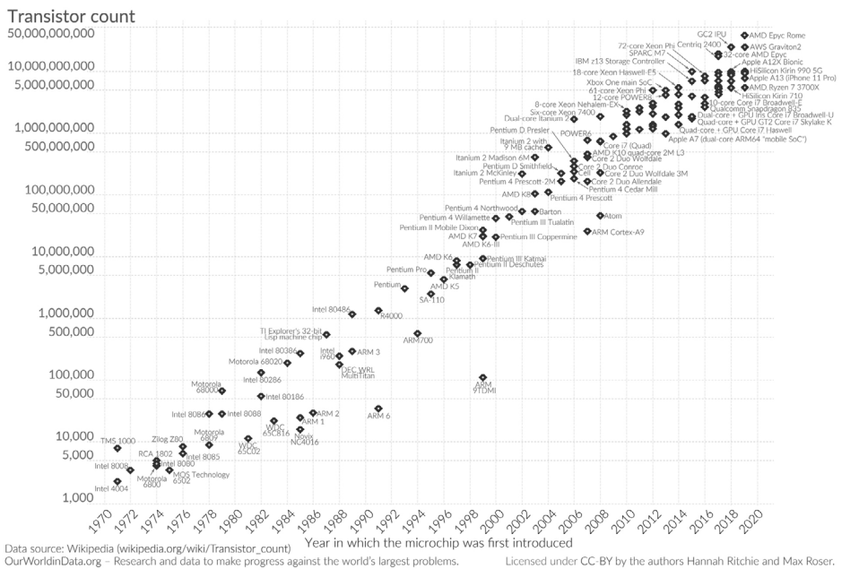
die momenteel gebruikt worden, hebben een actieve lengte van enkele nanometer, terwijl de grootste chips momenteel meer dan honderd miljard transistoren bevatten.[23] Deze trend wordt bevestigd door de grafiek van figuur 1.4, die historisch over de jaren heen het aantal transistoren in commercieel uitgebrachte processorchips weergeeft. De grafiek vertoont weliswaar een lineaire trend (in twee delen), maar de verticale schaal is logaritmisch! Dat betekent dat de rekenkracht van computers in werkelijkheid exponentieel[24] toeneemt met de tijd, wat natuurlijk met de jaren veel nieuwe toepassingen mogelijk gemaakt heeft voor een vaste kostprijs, of omgekeerd de kostprijs van bestaande toepassingen almaar goedkoper gemaakt heeft. Deze continue toename in bestaande en nieuwe toepassingen op een betaalbare wijze maakt ook dat elektronica, computers en de bijbehorende software een almaar verder reikende impact op onze samenleving verworven hebben. Merk op dat transistoren natuurlijk niet oneindig kleiner gemaakt kunnen worden; daarom is de nieuwste trend om meerdere siliciumchips in één verpakking naast of driedimensionaal boven elkaar te monteren teneinde dezelfde exponentiële groei in rekenkracht per eenheid basisoppervlakte te kunnen blijven realiseren als aangegeven in de wet van Moore.

Naast de rekenkracht per computer, is met de jaren ook het totale aantal computers dat gebruikt wordt sterk toegenomen. **Netwerken tussen computers** versterken namelijk de exponentiële groei in rekenmogelijkheden nog verder.[25] Dankzij die enorme toename in rekenkracht kunnen bestaande problemen natuurlijk sneller worden opgelost; in realiteit wordt de extra rekenkracht typisch gebruikt om almaar complexere problemen op te lossen of om almaar nieuwe en krachtigere toepassingen te ontwikkelen. Toepassingen die nu technisch nog niet haalbaar zijn (of die te lang duren of te veel energie vergen) zullen dat in de toekomst dus wel worden. Bovendien laat het netwerken toe om snel veel data te transfereren en om diensten aan te bieden waarvoor de rekenkracht of de dataopslag fysiek elders staan dan op je eigen computer. Daardoor kunnen nieuwe online-diensten worden aangeboden (bv. een zoekactie in een browser uitvoeren, audio of video streamen, een ChatGPT-opdracht starten, enz.). Daarvoor hoeft je lokale computer niet superkrachtig te zijn (bv. een eenvoudige laptop of tablet volstaat), terwijl de eigenlijke berekeningen elders gebeuren op een

[23] Zie voor meer informatie: en.wikipedia.org/wiki/Transistor_count.

[24] Bij een lineaire toename komt er altijd een constante hoeveelheid bij de huidige waarde bij, terwijl bij een exponentiële toename de huidige waarde met een constante factor wordt vermenigvuldigd: bijvoorbeeld maal 2, nogmaals maal 2, nogmaals maal 2, nogmaals maal 2, enzovoort, dus exponentieel t.o.v. de startwaarde.

[25] Zie over computernetwerken hoofdstuk 2 in dit boek.



Figuur 1.4: De wet van Moore: met de jaren neemt het aantal transistoren op een chip exponentieel toe (merk op dat de verticale as logaritmisch is). De grafiek toont de gegevens (aantal transistoren, jaar van introductie) van commercieel op de markt gebrachte processorchips.

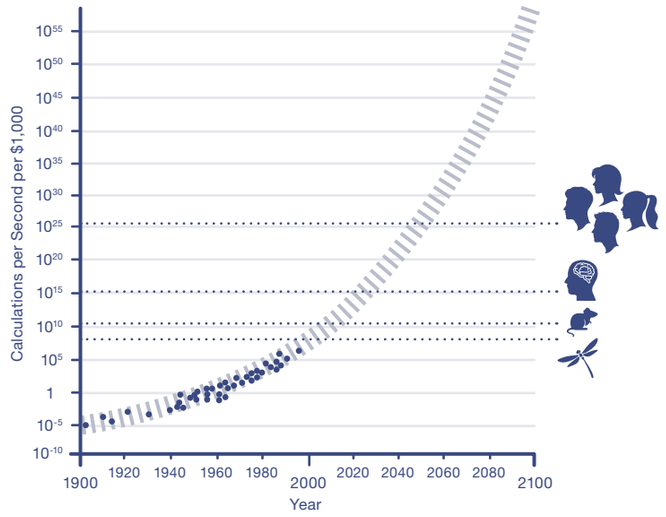
krachtige rekenserver[26] binnen je eigen organisatie of zelfs op een park van krachtige rekenservers in een extern datacenter; het resultaat daarvan wordt dan via het (internet)netwerk terug doorgestuurd naar je lokale computer (cliënt genoemd). De enorme toename in rekenmogelijkheden van genetwerkte computers verklaart bijgevolg de groeiende impact van softwaretoepassingen (en de bijbehorende industrietak) in onze samenleving, en dus ook het toenemende belang van algoritmes.[27] Anderzijds is er ook een toenemende kritiek op dit gebruiksmodel wegens het hoge energieverbruik van die datacenters, dat eigenlijk onmerkbaar is voor de eindgebruiker die er zich veelal niet van bewust is.

Merk op dat er over de jaren heen naast de gewone computers voor algemeen gebruik ook altijd **supercomputers** ontwikkeld werden. Dat zijn computers die speciaal ontworpen zijn voor hun buitengewoon groot rekenvermogen. Dat is mogelijk omdat

[26] Computerservers kunnen naast rekenserver ook voor veel andere specifieke diensten gebruikt worden, zoals fileservers, mailservers, webservers enzovoort, waarop iemand dan beroep doet vanop de lokale computer voor die specifieke taken.

[27] Zie sectie 3 in dit hoofdstuk.

ze intern veel processoren (i.p.v. één of een beperkt aantal processoren) bevatten die intern met een zeer snel netwerk verbonden zijn. Vaak is er ook veel geheugen- en opslagcapaciteit. Supercomputers kunnen ook worden gebouwd door meerdere computers te laten fungeren als één cluster; zelfs een groot gedistribueerd netwerk van veel gewone computers kan tegenwoordig als één supercomputer worden ingezet. Een supercomputer wordt bijvoorbeeld veelal gebruikt voor zware berekeningen in onderzoekswerk op universiteiten of onderzoeksinstellingen. Voorbeelden daarvan zijn berekeningen rond encryptie- en decryptiealgoritmes, kwantumfysica, weersvoorspelling, klimaatonderzoek, moleculaire modellering, aerodynamica, nucleaire fusie, enzovoort. Ook in Vlaanderen staat een supercomputer ter beschikking van het onderzoek van alle Vlaamse universiteiten via het Vlaamse Supercomputer Centrum (VSC).



Figuur 1.5: Door de exponentiële stijging in rekenkracht naderen we de technologische singulariteit. De afbeeldingen rechts geven symbolisch de equivalente rekenkracht van computers aan.

Wat gebeurt er nu als die exponentiële groei in rekenkracht blijft doorgaan? Daarvoor verwijzen we naar het boek '*The singularity is near*' van Ray Kurzweil (Kurzweil, 2005). De rekenkracht van computers kunnen we uitdrukken als het aantal berekeningen dat per seconde kan worden uitgevoerd per 1000 $ die we eraan besteden. Zoals de grafiek

van figuur 1.5 weergeeft, beslaat de wet van Moore voor CMOS-halfgeleiderelektronica maar een deel van de curve (van 1965 tot nu), en stijgt de rekenkracht feitelijk al exponentieel sinds het begin van de 20e eeuw, zij het initieel met andere technologieën dan halfgeleiders. In wezen is dat gewoon de weergave van het feit dat de mensheid steeds opnieuw de meest geavanceerde technologie gebruikt om almaar nieuwere en betere oplossingen te ontwikkelen, die dan weer ingezet worden om nog betere technologie te ontwikkelen, en zo almaar verder. Maar blijft dat zelfversterkend proces nog decades en eeuwen doorgaan? Ook als de huidige CMOS-schaalverkleining niet verder mogelijk is, omdat we de fysieke grenzen van het almaar verder verkleinen van de CMOS-transistoren naderen, is de verwachting dat wetenschappers en ingenieurs andere technologieën[28] ontwikkelen om de groei in rekenkracht verder te zetten. Dat wordt namelijk gedreven door de universele economische kracht van 'almaar meer voor minder'. Momenteel naderen computers een rekenkracht die (voor bepaalde taken) even krachtig is als het brein van een mens. Tegen 2040-2050 wordt verwacht dat computers even krachtig zullen zijn als de breinen van alle mensen samen, en vanaf dan worden computers dus krachtiger dan de mensheid. Men spreekt dan van de '**technologische singulariteit**',[29] een breekpunt in de tijd waarop de technologische vooruitgang zo snel zou kunnen gaan dat mensen met hun huidige intelligentie de resulterende maatschappij niet meer zouden kunnen begrijpen of beheersen. Dat breekpunt zou mogelijk kunnen worden gerealiseerd door zelflerende artificiële intelligentie. Doemdenkers zeggen dat dit mogelijk een risico inhoudt voor het voortbestaan van de mensensoort zoals we die nu kennen, tenzij we strikte voorwaarden of beperkingen opleggen aan wat die technologie in de toekomst mag doen en hoe die mag ingrijpen in onze leefwereld.[30]

2.4. TOEKOMSTIGE COMPUTERARCHITECTUREN

De exponentiële groei in rekenkracht vindt al meer dan een eeuw plaats, veel langer dan de wet van Moore bestaat. Om die continue groei over die lange periode technologisch te realiseren, waren/zijn er verschillende technologieën nodig: eerst (elektro)mechanische computers, dan computers opgebouwd met relais, vervolgens met halfgeleidende elektrische buizen, dan de huidige situatie met micro/nano-elek-

[28] Zie sectie 2.4 in dit hoofdstuk.

[29] Dit concept van technologische singulariteit werd voor het eerst uitvoerig beschreven door de Amerikaanse wiskundige en sciencefictionschrijver Vernor Vinge in zijn essay 'Singularity' (Vinge, 1993). Zie ook: https://www.youtube.com/watch?v=Vv_2C95cO6s.

[30] Dit is in essentie ook de boodschap in het boek 'Nexus' van Yuval Harari (Harari, 2024).

tronische halfgeleidertransistoren, en momenteel wordt er onderzoek gedaan naar nieuwe technologieën om in de toekomst nog krachtigere computers te realiseren. Mogelijkheden daarbij zijn neuromorfe computers, kwantumcomputers, fotonische computers, spintronicacomputers, DNA-gebaseerde computers, moleculaire computers, enzovoort. Welke van die technologieën uiteindelijk zullen doorbreken is op dit moment nog helemaal niet duidelijk: het hangt af van de performantie (rekensnelheid, energieverbruik) die kan worden gerealiseerd in elke technologie, maar ook van de kostprijs en de betrouwbaarheid e.d. waarmee ingenieurs die computers kunnen bouwen. Hieronder bespreken we bij wijze van voorbeeld kort twee zeer beloftevolle toekomstige technologieën, namelijk neuromorfe[31] en kwantumcomputers,[32] met hun mogelijkheden en beperkingen.

2.4.1. Neuromorfe computers

Klassieke elektronische computers die gebaseerd zijn op de von Neumann-architectuur zoals hierboven is beschreven, zijn zeer geschikt voor de uitvoering van numerieke berekeningen (bv. de evolutie van het weer voorspellen via het numeriek uitrekenen van de wiskundige modellen die het weer beschrijven). Maar ze zijn veel minder efficiënt in de uitvoering van cognitieve taken (bv. de herkenning en classificatie van personen of dieren op foto's). Daarvoor wordt gekeken naar **neuromorfe computers**, d.w.z. computers die architecturaal zo zijn opgebouwd dat ze de neurobiologische architectuur en werking van onze hersenen en zenuwen proberen na te bootsen of te emuleren. Een belangrijke eigenschap daarbij is dat het – net zoals het menselijk brein – (zelf)lerende systemen zijn die via training en continue bijstelling voortdurend hun werking kunnen verbeteren en die zich kunnen aanpassen aan de omstandigheden of aan veranderende situaties (we noemen dat plasticiteit). Ze kunnen beslissingen nemen op basis van gegevens die ze bijvoorbeeld ontvangen van externe sensoren (de 'zintuigen' van de computer) en ze leren patronen te herkennen in die gegevens. Dergelijke 'artificiële neurale systemen' die geïnspireerd zijn op ons menselijk brein en die getraind worden om patronen te herkennen uit data, worden dus vooral gebruikt voor cognitieve toepassingen zoals visie, gehoor, autonome robots, autonome auto's, en dergelijke. Een goed begrip van hoe ons brein precies werkt is daarbij uiterst belangrijk; en alhoewel we op dat vlak al heel wat vooruitgang hebben geboekt, is daarover toch nog niet alles geweten. Merk op dat het niet de bedoeling is om het brein exact na te bouwen, maar wel om uit de structuur en de werking van het brein

[31] Zie sectie 2.4.1 in dit hoofdstuk.
[32] Zie sectie 2.4.2 in dit hoofdstuk.

inspiratie te halen om een praktisch computersysteem te bouwen voor de beoogde toepassingen. Kenmerkend is dat een dergelijk computersysteem typisch bestaat uit een gedistribueerd netwerk van relatief eenvoudige rekenelementen (gelijkaardig aan onze neuronen).[33] Vanzelfsprekend kan men dergelijke artificiële neurale systemen ook op klassieke bestaande computers in software programmeren, maar implementaties in specifieke neuromorfe hardware zijn veelal sneller en verbruiken minder energie. Al in 2019 heeft de Vlaamse onderzoeksinstelling imec een zelf-lerende neuromorfe chip gerapporteerd die muziek componeert. Recente voorbeelden van neuromorfe processoren zijn Intel's Loihi 2 chip (die tot 1 miljoen neuronen en 120 miljoen synapsen ondersteunt) (2021) en IBM's NorthPole chip (2023). Hardwarematig worden er verschillende alternatieve technologieën onderzocht om de neuromorfe systemen efficiënt te implementeren. Het domein is dus in voortdurende snelle evolutie en er worden nog veel krachtigere neuromorfe computerchips verwacht in de komende jaren.

2.4.2. Kwantumcomputers

Voor snellere berekeningen wordt in de nabije toekomst veel verwacht van **kwantumcomputers**. Dat zijn computers waarbij de rekenprocessor (de *Quantum Processing Unit* of QPU genoemd) gebruikmaakt van de principes van de kwantummechanica (d.w.z. superpositie, verstrengeling en interferentie van deeltjes of fotonen). Het basiselement van de QPU zijn qubits (quantum bits), die kwantummechanisch tegelijkertijd de waarden '0' en '1' kunnen hebben (in tegenstelling tot de bits bij klassieke computers die alleen '0' of '1' kunnen zijn); berekeningen met qubits worden dus tegelijk voor beide waarden 0 en 1 uitgevoerd. Daardoor stijgt de rekenkracht exponentieel met het aantal qubits ($\sim 2^{\#\text{qubits}}$). Een kwantumprocessor kan dus in parallel, d.w.z. tegelijk, dezelfde berekeningen uitvoeren over een zeer grote hoeveelheid data, en is daardoor – tenminste voor specifieke problemen die daarvoor geschikt zijn – intrinsiek veel sneller dan klassieke computers, die de berekeningen herhaaldelijk moeten uitvoeren voor alle data. Als toepassingen denkt men aan berekeningen op het domein van encryptie (cyberbeveiliging van gegevens), het snel zoeken in grote databanken, kwantumsimulaties van bijvoorbeeld chemische reacties, kunstmatige intelligentie en andere vormen van machinaal leren, enzovoort.

Een beperking tot op dit moment is dat de verstrengeling van qubits vrij snel onstabiel wordt (decoherentie genoemd), waardoor op dit moment het aantal betrouwbaar uit te voeren kwantumberekeningen en dus de lengte van de uit te voeren kwantum-

[33] Zie verder hoofdstuk 4 in dit boek.

algoritmes in de praktijk nog vrij beperkt is. Er wordt dan ook hard gezocht naar goede foutencorrectiemethodes. De huidige kwantumcomputers zijn nog beperkt in aantal qubits (bv. IBM's Condor kwantumprocessor met 1121 qubits werd eind 2023 aangekondigd) – voor praktische toepassingen zullen wellicht computers met vele duizenden fysische qubits nodig zijn. Bovendien moeten ze nog altijd extreem worden gekoeld tot temperaturen dicht bij het absolute nulpunt. In 2019 beweerde Google dat ze kwantumsuprematie gerealiseerd hadden (d.w.z. dat ze met een kwantumcomputer in een korte tijdspanne een probleem hadden opgelost dat in geen realistisch eindige tijdspanne met een klassieke supercomputer op te lossen is), maar IBM heeft dat bericht nadien ontkracht.[34] In december 2020 berichtte een onderzoeksteam uit China dat ze kwantumsuprematie bereikt hadden voor het wiskundig-natuurkundig probleem van gaussiaanse bosonbemonstering op een door hen gebouwde fotonische kwantumcomputer.[35] Er verschijnen nog regelmatig andere aankondigingen en er zullen er wellicht nog vele volgen.

Het domein van kwantumcomputing is dus in volle evolutie, zowel op het vlak van de kwantumhardwaretechnologie als op het vlak van de kwantumsoftwarealgoritmes. Volgens de **voorspelling van Hartmut Neven**, directeur van het Quantum AI Lab van Google, uit 2018 neemt de rekenkracht van kwantumcomputers met de tijd toe volgens een dubbel-exponentiële functie. Die stijgt dus nog veel sneller dan een gewone exponentiële functie,[36] maar er zijn nog onvoldoende datapunten om dat ook effectief op langere termijn te bevestigen. Interessant in die context zijn de kwantumroadmaps van firma's zoals IBM of Google, die je makkelijk online vindt. Ook wordt er momenteel gewerkt aan kwantumcentrische supercomputers waar klassieke en kwantumcomputers gecombineerd worden om de voordelen van beide te gebruiken.

3. ALGORITMES

Het grote voordeel van computers is dat ze niet beperkt zijn tot één specifieke taak, maar dat ze programmeerbaar zijn om meerdere taken uit te voeren met dezelfde hardware. Dat programmeren gebeurt met een softwareprogramma, dat geschreven

[34] Zie een interessante MIT-podcast over het getouwtrek tussen Google en IBM over kwantumsuprematie op: www.technologyreview.com/2020/02/26/905777/google-ibm-quantum-supremacy-computing-feud/.
[35] Zie voor meer informatie: en.wikipedia.org/wiki/Quantum_supremacy.
[36] Zie voor meer informatie over de wet van Hartmut Neven: en.wikipedia.org/wiki/Hartmut_Neven#Neven's_law.

wordt in een programmeertaal die voor de programmeur of informaticus handig en begrijpbaar is om mee te werken (bv. Java, C++, Python, ...). Dat programma (de broncode genoemd) is dus typisch computeronafhankelijk. Vervolgens wordt dat softwareprogramma vertaald (gecompileerd) naar een uitvoerbaar of executeerbaar programma (de uitvoeringscode genoemd), dat uitgeschreven is in termen van de specifieke instructies die gekend zijn door de computer die je wenst te gebruiken, inclusief kennis van de beschikbare geheugens, invoer en uitvoer. Door ten slotte die uitvoeringscode effectief uit te voeren op de hardware van de computer, realiseert de computer de gewenste acties naar het beoogde einddoel toe.

Om met de computer een taak op te lossen wordt veelal een algoritme[37] uitgevoerd. In de volgende secties leggen we eerst uit wat een algoritme is[38] om vervolgens de kenmerken ervan te bespreken.[39] We gaan vervolgens in op de classificatie van algoritmes[40] en eindigen met een concrete illustratie van een algoritme.[41]

3.1. WAT IS EEN ALGORITME?

Een **algoritme** is een eindige reeks instructies om een (al dan niet wiskundig) probleem op te lossen, d.w.z. een procedure om stapsgewijs het probleem van een gegeven begintoestand naar een beoogd einddoel te brengen.

Een algoritme is dus vergelijkbaar met een recept om stapsgewijs een gerecht te bereiden. Voorbeelden zijn algoritmes voor het tekenen van een cirkel op een computerscherm, het numeriek berekenen van de oplossingen van een veeltermvergelijking van hogere orde, het bepalen van de resultaten van een zoekactie in een webbrowser, enzovoort. Het op te lossen probleem is vaak wiskundig gedefinieerd. Dikwijls zijn er ook meerdere manieren mogelijk om het probleem op te lossen, en zijn er dus ook meerdere algoritmes voor hetzelfde probleem mogelijk. Ze verschillen meestal in de

[37] De term algoritme is afgeleid van de naam van de Perzische wiskundige Musa Al-Chwarizmi uit de 9e eeuw.
[38] Zie sectie 3.1 in dit hoofdstuk.
[39] Zie sectie 3.2 in dit hoofdstuk.
[40] Zie sectie 3.3 in dit hoofdstuk.
[41] Zie sectie 3.4 in dit hoofdstuk.

hoeveelheid tijd en/of hardware die nodig is om het probleem op een computer op te lossen (men noemt dat de rekencomplexiteit of efficiëntie van het algoritme).

Terwijl een algoritme eerder een oplossingsmethode is die strikt genomen losstaat van computerprogramma's, worden in de praktijk meestal computers gebruikt voor de uitvoering van algoritmes. Daar waar een algoritme dus de beschrijving is van de oplossingsprocedure voor een probleem, is het overeenkomstige computerprogramma (geschreven in een of andere programmeertaal) de implementatie van dat algoritme. Bij de uitvoering van het computerprogramma is het belangrijk dat het algoritme effectief de juiste taak uitvoert en het correcte resultaat genereert, en dat het dat in een aanvaardbare tijd doet.

Door de toenemende rekenkracht van computers neemt het aantal en de mogelijkheden van de softwareprogramma's in onze samenleving toe. Daardoor kunnen almaar krachtigere algoritmes en hun software-implementaties worden gerealiseerd. Tegelijk moeten ook de ethische en juridische implicaties van die software goed in het oog worden gehouden. Zo kun je software om foto's te bewerken gebruiken om bijvoorbeeld een toevallig puistje op je gezicht weg te werken, maar kunnen anderen ook je foto manipuleren en misbruiken in een verkeerde context (bv. deepfakes creëren). Hoe krachtiger de mogelijkheden van software, hoe moeilijker het veelal is om echt van fake en bonafide van malafide te onderscheiden. Een ethisch bedenkelijk voorbeeld is het softwarematig afslanken van je gezicht, dat een verkeerd ideaalbeeld kan creëren en anderen kan aanzetten tot zelfuithongering of tot plastische chirurgie om ook hun gezicht daadwerkelijk afgeslankt te krijgen (Hunt, 2019). Een andere bezorgdheid is hoe men moet omgaan met algoritmische vooringenomenheid (*bias* genoemd), het fenomeen waarbij slimme algoritmes geneigd zijn om bevooroordeelde beslissingen te nemen, ook al is dat veelal niet bewust intentioneel. Dat komt bijvoorbeeld omdat de gebruikte of beschikbare trainingsdata niet volledig representatief zijn voor de populatie waarover de algoritmes uitspraak zouden moeten doen. Verschillende auteurs, zoals Cathy O'Neil (O'Neil, 2016), waarschuwen dan ook voor de mogelijke schaduwkanten van het alomtegenwoordig en vooral het blindelings gebruik van algoritmes (Brandom, 2018; Simonite, 2018).[42]

Algoritmes bestaan typisch uit meerdere stappen in sequentie, die zich kunnen herhalen (dat noemen we iteratie) of die beslissingen (met behulp van logica of vergelijkingen) vereisen om de taak te voltooien. Een simpel voorbeeld is het algoritme

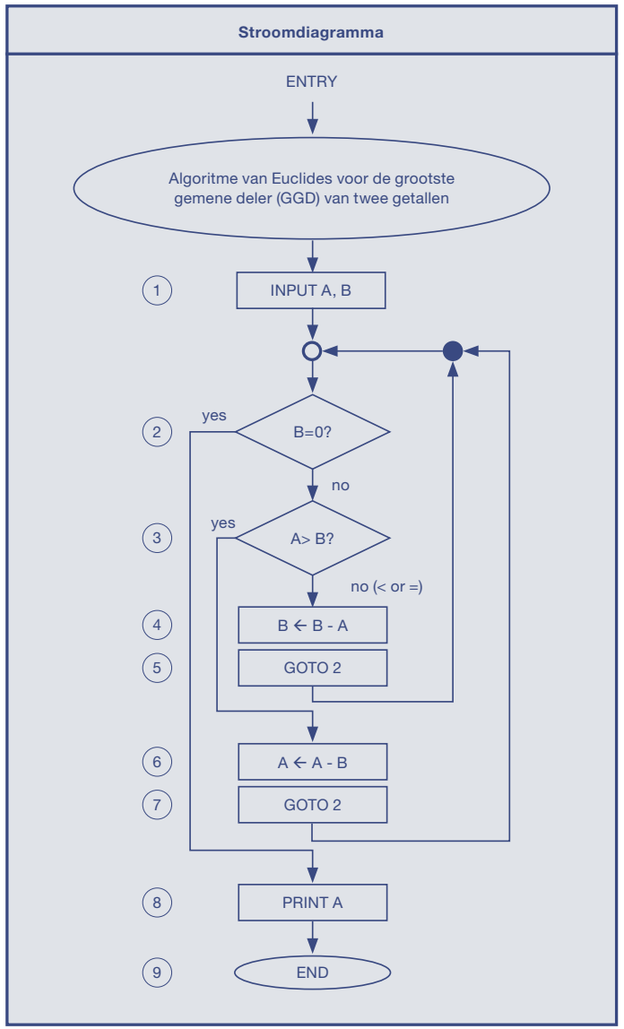
[42] Zie uitvoerig over deze problematiek in hoofdstuk 3 in dit boek.

van Euclides om de grootste gemene deler[43] van twee getallen a en b te bepalen. Dat algoritme is hieronder beschreven. Het gebruikt de twee wiskundige hulpvariabelen A en B (initieel gelijk aan de ingevoerde getallen a en b). Veelal wordt een algoritme voorgesteld als een stroomdiagramma (zie figuur 1.6), maar een wiskundige formulering of een formulering in pseudo-computercode zoals hieronder is even handig.

Algoritme van Euclides om de grootste gemene deler van twee getallen a en b te bepalen:
initieel: A = a en B = b;
als B = 0: uitkomst is A en stop;
als A > B:
    dan: trek B af van A en ga terug naar de vorige stap;
    anders: trek A af van B en ga terug naar de vorige stap.

Een voorbeeld van dit algoritme van Euclides toegepast op de getallen a = 36 en b = 63 vind je hier. Initieel is A = a = 36 en B = b = 63. Zolang B niet 0 is, blijven we dezelfde stappen herhalen. In de eerste iteratie is A < B, dus volgen we het pad 'anders:' en trekken we A af van B, waardoor B aangepast wordt naar B = 27. In de tweede iteratie is A nu groter dan B, volgen we het pad 'dan:', wordt B afgetrokken van A en wordt A = 9. In de derde iteratie is B opnieuw groter dan A en wordt B aangepast naar B = 18. In de vierde iteratie is B nog altijd groter dan A en wordt B aangepast naar B = 9. In de vijfde iteratie is A niet groter dan B en wordt B aangepast naar B = 0. Aangezien B nu 0 is, wordt als resultaat A = 9 doorgegeven als zijnde de grootste gemene deler van a = 36 en b = 63 en stopt het algoritme.

[43] De grootste gemene deler (GGD) van twee getallen is het grootste positieve gehele getal waar beide gehele getallen door gedeeld kunnen worden zonder dat er een rest overblijft.



Figuur 1.6: Stroomdiagramma van het algoritme van Euclides om de grootste gemene deler van twee getallen te bepalen.

De procedure van een algoritme is vanzelfsprekend gebaseerd op kennis over de wereld. Daarbij maken we een onderscheid tussen declaratieve en procedurele kennis. Declaratieve kennis is een feitelijke uitspraak over een aspect uit de wereld. Het geeft met andere woorden de definitie van een probleem. Bijvoorbeeld is de wiskundige uitdrukking voor de punten die in het tweedimensionale (x, y)-vlak op een cirkel met middelpunt (a, b) en straal r liggen als volgt: $(x - a)^2 + (y - b)^2 = r^2$. Maar dat zegt niets over hoe je een dergelijke cirkel op een computerscherm moet tekenen. Daarvoor heb je procedurele kennis nodig. Die geeft een stapsgewijze methode van oplossing. Bijvoorbeeld: je tekent vanuit een startpunt op het scherm een klein lijntje vooruit, dan draai je over een klein aantal graden en teken je weer een klein lijntje vooruit, enzovoort. Als je voldoende kleine lijnstukjes met heel kleine stapjes en draaihoekjes tekent wordt dat uiteindelijk een cirkel op je scherm. Voor een algoritme hebben we een stapsgewijze of procedurele oplossingsmethode nodig.

3.2. KENMERKEN VAN ALGORITMES

Aan welke vereisten moet een algoritme voldoen? Sommige eigenschappen zijn eerder belangrijk voor de gebruiker, andere dan weer eerder voor de informatici en ingenieurs die het algoritme programmeren. De ontwikkelaars van de softwareprogramma's moeten met al die eisen rekening houden.

Om bruikbaar te zijn moet een algoritme aan volgende kenmerken voldoen:

1. *correctheid*: het algoritme moet correct zijn, d.w.z. de juiste uitvoer geven voor een geldige invoer; merk op dat de uitkomst niet altijd exact kan zijn maar wel benaderend juist (bv. de berekening van de waarde van het getal π kan maar benaderend tot op een aantal cijfers na de komma worden berekend).
2. *eindigheid*: het algoritme moet in een eindige tijd stoppen met het correcte antwoord; het mag niet eindeloos blijven doordraaien.
3. *uitvoeringssnelheid of efficiëntie*: hoe snel berekent het algoritme het gewenste antwoord? Belangrijk daarbij is de rekencomplexiteit van het algoritme, d.w.z. hoeveel neemt de benodigde uitvoeringstijd toe als de grootte van de taak toeneemt (bv. bij een algoritme dat een reeks opgegeven getallen volgens grootte moet sorteren: hoe snel neemt de rekentijd toe als het aantal getallen toeneemt).

4 *robuustheid*: kan het algoritme overweg met alle mogelijke vormen van invoer, zelfs met onverwachte of foutieve invoer? (bv. wat doet een programma om de vierkantswortel van een (positief) getal te berekenen, als men een negatief getal ingeeft of iets ingeeft dat helemaal geen getal is?)

Voor de algoritmeontwikkelaars en programmeurs zijn ook de volgende kenmerken belangrijk:

1 *elegantie*: is het algoritme (en de computercode) leesbaar en begrijpbaar?
2 is een algoritme *ethisch en zonder vooringenomenheid*? Zeker als we de resultaten van algoritmes zonder meer gebruiken of als we algoritmes autonoom beslissingen laten nemen (bv. een algoritme dat autonoom cv's scant en een preselectie maakt van de kandidaten voor een jobvacature).
3 is een algoritme *duurzaam*? Wat is de impact op het milieu? (bv. een simpele zoekactie in een internetbrowser heeft toch een sterke ecologische voetafdruk omdat ze elders ter wereld een serverpark hard laat werken om de zoekresultaten te berekenen).

En tot slot zijn er natuurlijk nog eisen vanwege de eindgebruiker: het computerprogramma dat het algoritme implementeert, moet eenvoudig te gebruiken zijn (gebruiksvriendelijkheid), een gebruikersinterface hebben die er grafisch mooi uitziet, niet te duur zijn in aanschafprijs, upgrades aanbieden als er fouten in het programma gedetecteerd worden, enzovoort.

3.3. CLASSIFICATIE VAN ALGORITMES

Algoritmes kunnen op verschillende manieren worden geclassificeerd. Voor de context van dit boek is de volgende indeling belangrijk:

- **Deterministische algoritmes**: deze algoritmes produceren voor een gegeven invoer altijd dezelfde uitvoer, waarbij het algoritme altijd dezelfde sequentie van stappen doorloopt. Een voorbeeld is bovenstaand algoritme van Euclides om de grootste gemeenschappelijke deler van twee getallen te berekenen.

- **Probabilistische of stochastische algoritmes**: dit zijn algoritmes die, zelfs voor dezelfde invoer, een verschillende uitkomst kunnen genereren, afhankelijk van een willekeurig (*'random'*) proces dat bewust in het algoritme wordt gebruikt. Een voorbeeld is een algoritme voor het bepalen van het optimum van een wiskundige functie, dat vertrekt vanuit een willekeurig startpunt of dat op willekeurige (*random*) wijze opeenvolgende stappen genereert om het optimum te zoeken. Daardoor kunnen opeenvolgende uitvoeringen van het algoritme tot verschillende oplossingen leiden. Veelal worden dergelijke niet-deterministische algoritmes gebruikt om een goede benaderende oplossing te verkrijgen als de rekentijd om de exacte oplossing (bv. het exacte optimum) m.b.v. deterministische algoritmes te berekenen veel te lang zou zijn – in de praktijk is een goede benaderende oplossing in een redelijke tijd verkrijgen namelijk veelal voldoende.

- **Slimme algoritmes of machinaal leren (ML)**: dit zijn algoritmes die zichzelf automatisch verbeteren op basis van vroegere ervaringen. Hoe meer het algoritme traint met nieuwe gegevens (data), hoe beter het algoritme (meestal) wordt. Daartoe bouwt het algoritme een model van het probleem op op basis van de trainingsdata, en vervolgens gebruikt het dat model om de uitkomst te bepalen voor nieuwe invoer. Machinaal leren behoort tot het bredere domein van de kunstmatige of artificiële intelligentie.[44]

3.4. ILLUSTRATIE: ALGORITMES VOOR HET 'RAAD-MIJN-GETAL'-SPELLETJE

Laat ons deze begrippen over algoritmes nu illustreren toegepast op het spelletje '*raad mijn getal*': een eerste persoon (bv. Jan) kiest in zijn hoofd een geheel getal tussen 1 en 128 (in het algemeen tussen 1 en N) en een tweede persoon (bv. Mieke) moet een efficiënte methode bedenken om dat getal te raden. Er zijn verschillende mogelijke oplossingswijzen of algoritmes om dat op te lossen. Elk algoritme stopt als het getal geraden is. Voor elk algoritme onderzoeken we of het correct is, of het eindig is en wat de (gemiddelde) uitvoeringssnelheid is in functie van het aantal mogelijke getallen N.

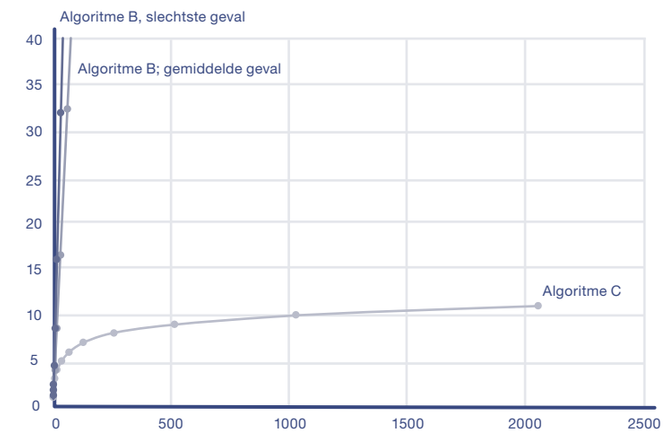
- ***Algoritme A:*** Mieke raadt willekeurig een getal en raadt telkens opnieuw als het getal fout is.
  → Dit algoritme is niet efficiënt en stopt mogelijk nooit (als Mieke niet bijhoudt welke getallen ze al geprobeerd heeft).

[44] Zie een uitgebreide beschrijving in hoofdstuk 3 in dit boek.

- ***Algoritme B:*** Mieke raadt sequentieel alle getallen startende van 1 opwaarts of van N neerwaarts.
  → Dit algoritme stopt gegarandeerd met de juiste oplossing, maar in bovenstaand voorbeeld van 128 getallen zijn daarbij gemiddeld 64,5 pogingen nodig. In het algemeen geval met N getallen zijn er (N + 1)/2 pogingen nodig, als gemiddelde tussen het snelste geval (het getal raden in 1 keer) en het traagste geval (men moet N keer raden om het getal te vinden). De rekencomplexiteit is dus lineair met het aantal getallen N.

- ***Algoritme C:*** Mieke raadt willekeurig een getal waarop Jan zegt of het getal in zijn hoofd hoger of lager is dan dat getal; dan geeft Mieke een nieuw getal en zegt Jan weer of het getal hoger of lager is dan dat tweede getal, en zo verder.
  → Ook deze methode is eindig en leidt tot de juiste oplossing. Mieke kan bijvoorbeeld de zoekruimte telkens halveren door het middelste getal op te geven van het resterende getalinterval waarbinnen het gekozen getal ligt (dat interval zelf is natuurlijk afhankelijk van de antwoorden van Jan). In bovenstaand voorbeeld zou Mieke bijvoorbeeld achtereenvolgens 64, 32, 48, 40 enzovoort kunnen kiezen als het getal bijvoorbeeld 43 zou zijn.
  → Inzake uitvoeringssnelheid moet Mieke maximaal 7 keer raden om tot een zoekruimte van 1 getal (het juiste getal) te komen. Vanwege de opeenvolgende halveringen van de zoekruimte moet men in het algemeen geval $\log_2(N)$ keer raden bij N getallen. De efficiëntie van algoritme C is dus logaritmisch met N, wat al stukken sneller is dan het lineaire algoritme B.

De grafiek in figuur 1.7 geeft op de verticale as de uitvoeringssnelheid (= het aantal benodigde stappen) weer van bovenstaande algoritmes B en C in functie van het aantal getallen N (op de horizontale as). Zoals men kan zien is algoritme C veel sneller dan het gemiddelde geval en zeker dan het slechtste geval van algoritme B. De eerder trage toename van de rekentijd met het aantal getallen is precies wat ingenieurs en programmeurs zoeken bij de ontwikkeling van algoritmes. Naast het gebruiksgemak bepaalt deze rekencomplexiteit (d.w.z. hoe de rekentijd toeneemt met de grootte van het probleem) ook vaak of een softwareproduct commercieel haalbaar is of niet.



Figuur 1.7: Rekencomplexiteit van algoritme B en C in het 'raad-mijn-getal'-spelletje in functie van het aantal mogelijke getallen.

Bij wijze van voorbeeld is in de linkerkolom van onderstaande tabel de softwarecode van algoritme C in pseudocode weergegeven. In de rechterkolom is datzelfde algoritme in de programmeertaal Python weergegeven (Python is hier louter een voorbeeld; er zijn nog vele andere programmeertalen).

Pseudocode	Python

getal = ... (te kiezen door Jan) laag = 1;hoog = 128;
 	
zolang getal niet geraden is:
 	
"Is het lager (incl. gelijk) of hoger dan ((laag+ hoog)/2)?"
=> indien lager (of gelijk): hoog = (laag+ hoog)/2
=> indien hoger: laag = ((laag+ hoog)/2) + 1
Herhaal (tot het geraden is)
 	
spel afgelopen
	def hogerLagerSpel (getal):
	
	laag = 0
	hoog = 128
	while laag < hoog
	midpoint = (laag + hoog)/2 if te_raden_getal <=
	midpoint:
	hoog = midpoint
	else:
	laag = midpoint + 1
	return laag

Tabel 1.1: Illustratieve pseudocode (links) en Python-code (rechts) van algoritme C van het 'raad-mijn-getal'-spelletje.

De vraag is nu of we nog sneller kunnen? Dat brengt ons bij het 'slimme' algoritme D.

- ***Algoritme D:*** veronderstel dat Mieke kennis heeft van alle in het verleden door Jan gekozen getallen (zijn voorkeurgetallen dus), dan kan algoritme C op basis van die kennis als volgt worden aangepast: in plaats van de resterende zoekruimte telkens gewoon gelijk in twee te splitsen, kan Mieke de zoekruimte zo splitsen dat er evenveel voorkeurgetallen onder als boven het splitsgetal liggen. Dat noemen we een 'slim' algoritme, omdat er gebruik gemaakt wordt van de kennis van vorige edities waarin ze het spel gespeeld hebben. Naarmate het spel meer en meer gespeeld wordt en Mieke meer informatie heeft over de voorkeurgetallen van Jan, kan het algoritme dus continu worden aangepast om sneller het resultaat te verkrijgen. Het algoritme kan ook zichzelf aanpassen als het de gekozen getallen uit het verleden systematisch bijhoudt en op basis daarvan de keuze van zijn splitsgetallen aanpast.
  - → Ook deze methode is eindig en leidt tot de juiste oplossing, want Mieke verkleint altijd het resterende interval tot er 1 getal (het juiste getal) overblijft.
  - → Inzake uitvoeringssnelheid zal dit gemiddeld sneller zijn dan algoritme C, omdat men gebruik maakt van de informatie over de voorkeurgetallen van Jan. Maar soms kan het langer duren dan algoritme C, namelijk als Jan afwijkt van zijn historisch voorkeurgedrag.

Er zijn momenteel al veel toepassingen van 'slimme' algoritmes in gebruik. Een voorbeeld is de zoekmodule in een internetbrowser: via de links en/of *likes* waarop je wel of niet geklikt hebt in het verleden wordt de volgorde van weergave van de resultaten van een nieuwe zoekactie mee bepaald. Ook de routebepaling van Waze maakt gebruik van de informatie van andere weggebruikers die al voor jou dat traject gevolgd hebben om de snelste route naar je bestemming te bepalen. Nog een voorbeeld zijn de aanbevelingen die je krijgt bij online shoppingsites, die mee bepaald worden door je voorafgaand zoekgedrag en vooral je voorafgaand aankoopgedrag.

In het algemeen maken technieken van **machinaal leren** (*machine learning* of afgekort ML) en van **kunstmatige of artificiële intelligentie** (afgekort AI) zoveel mogelijk gebruik van beschikbare gegevens (data) om de werking en de doelmatigheid van de algoritmes zelf te verbeteren. De doorbraak van dergelijke slimme algoritmes die we momenteel in onze samenleving vaststellen heeft enerzijds te maken met de vooruitgang in computers die voldoende rekenkrachtig geworden zijn met voldoende grote

geheugens, en anderzijds omdat er nu massaal veel gegevens digitaal beschikbaar komen. Vandaar ook de bewering 'Data is de nieuwe olie'.[45]

Tot slot nog een pertinente vraag voor slimme algoritmes die getraind worden met grote hoeveelheden data (die voortdurend kunnen wijzigen en zich aanpassen of die kunnen verschillen van gebruiker tot gebruiker): wie is er uiteindelijk verantwoordelijk of aansprakelijk voor het gedrag en de resultaten van een slim algoritme? Zijn het de verkopers of de aanbieders van de software, de softwareprogrammeurs, de gebruikers die het algoritme trainen of wijzigen met behulp van eigen data, enzovoort? Dit wordt in een later hoofdstuk besproken.[46]

4. SOFTWARE

Terwijl we in de vorige secties dieper ingingen op ICT-hardware, staat in de volgende secties de software centraal. We bekijken ten eerste hoe een softwareprogramma in het algemeen tot stand komt.[47] We gaan ook dieper in op het verschil tussen broncode en uitvoeringscode[48] en tot slot bespreken we het concept van openbronsoftware.[49]

4.1. SOFTWAREONTWIKKELING

Softwareontwikkeling is een productieproces door verschillende groepen programmeurs. Zeker voor grotere, meer complexe software zoals die in de praktijk gebruikt wordt (bv. YouTube, Google Chrome, Powerpoint, enz.) zijn er grote groepen programmeurs nodig. Typisch bestaan softwareprogramma's tegenwoordig uit verschillende 'lagen' softwarecode bovenop elkaar, veelal door verschillende mensen en bedrijven ontwikkeld. Elke laag software maakt dan gebruik van softwareroutines van de laag eronder. Een programma zoals de Google-zoekmachine wordt door de gebruiker ervaren als één groot programma dat één algoritme uitvoert, maar is feitelijk opgebouwd uit honderden kleine stukjes code, die elk een algoritme op zich zijn. Deze

45 Uitspraak van Clive Humby, Chief Data Scientist van de firma Starcount (en.wikipedia.org/wiki/Clive_Humby). Zie hoofdstuk 3 voor meer informatie over deze technieken.
46 Zie hoofdstuk 6 in dit boek voor meer informatie.
47 Zie sectie 4.1 in dit hoofdstuk.
48 Zie sectie 4.2 in dit hoofdstuk.
49 Zie sectie 4.3 in dit hoofdstuk.

modulaire opbouw maakt de softwareontwikkeling technisch haalbaarder, maar maakt de software tegelijk ook moeilijker te doorgronden, en het is moeilijker te achterhalen welk deel van de software precies welke beslissingen neemt of waar precies een softwarefout zit.

Bij de ontwikkeling van een softwareprogramma moeten de programmeurs natuurlijk ervoor zorgen dat de software de gewenste functionaliteit heeft en dat ze voldoet aan de vereisten. Deze specificaties worden afgesproken bij de start van de ontwikkeling. Daarbij moet rekening gehouden worden met een groot aantal randvoorwaarden:

1 Wie is de eindgebruiker: andere programmeurs of de (interne of externe) eindgebruiker, of wordt de software ingebed in een apparaat?
2 Is de software bedoeld voor openbrongebruik (d.w.z dat de broncode van de software voor het brede publiek beschikbaar is en de software door iedereen (veelal gratis) gebruikt kan worden) of voor propriëtair gebruik (d.w.z. dat de gebruikers (meestal tegen betaling) alleen een licentie op een uitvoerbare versie verkrijgen)?[50]
3 Wat is de vereiste veiligheid of de gewenste graad van zekerheid over een correcte uitvoering van de software? Software die nooit mag crashen is veel moeilijker te ontwikkelen en te debuggen (zie de volgende sectie). In sommige toepassingen (bv. een streaming app) is het vervelend maar maakt het minder uit, in andere (bv. de besturing van een auto) juist wel en zijn fouten eigenlijk niet toegelaten.
4 Wat zijn de vereisten in termen van uitvoeringssnelheid?
5 Op welke types van machines moet de software draaien?
6 Enzovoort

4.2. SOFTWARECODE EN FOUTEN (*BUGS*)

Software wordt typisch geschreven in broncode (*source code*) in een voor programmeurs leesbare programmeertaal (bv. Python, C, C++, enz.). Die broncode is eigenlijk onafhankelijk van de gebruikte computer, maar wordt vervolgens door een compileerprogramma (*compiler*) gecompileerd of omgezet in uitvoeringscode (*execution code*) die specifiek is voor de computer die gebruikt wordt. Die uitvoeringscode is binair (enen en nullen) en kan door de gebruikte computer worden begrepen en uitgevoerd.

[50] Zie voor meer details hierover sectie 4.3 in dit hoofdstuk.

Merk op dat het voor juridische disputen de broncode is die relevant is. Die broncode wordt tegenwoordig veelal ook grotendeels automatisch gegenereerd door applicatiesoftware gericht op het maken van programma's. Neem als voorbeeld de internethomepagina van een krant. Als je naar de broncode van die pagina in de HTML-taal gaat kijken,[51] zul je snel begrijpen dat die broncode natuurlijk niet allemaal manueel ingetikt kan zijn door een programmeur. Integendeel, de grafische ontwerper van de homepagina van de krant gebruikt een layoutprogramma voor webpagina's om snel op grafische wijze die homepagina te creëren. Het webpaginaontwerpprogramma genereert vervolgens automatisch de overeenkomstige HTML-code, die dan door je webbrowser wordt omgezet in de grafische webpagina die je op je scherm ziet als je de homepagina oproept.

Er is een grote variatie aan softwareprogramma's. Naargelang de functie die de software vervult, kunnen we volgende verschillende types van software onderscheiden:

- programmeertalen (bv. Python, Java, C, ...);
- besturingsystemen (*operating systems*) (bv. Windows, Linux, Android, ...);
- communicatieprotocollen (bv. TPC/IP, ftp, ...);
- toepassingen of applicaties (bv. apps, spelletjes, tekstverwerker, browser, ...).

Onvermijdelijk bevatten softwareprogramma's altijd fouten waardoor ze hun functie niet (geheel) volgens de vereisten of de specificaties vervullen. Dergelijke fouten worden vanuit het Engels softwarebugs genoemd. Commerciële software heeft typisch een groot aantal bugs, dat van de orde van 20 tot 30 bugs per 1000 lijnen softwarecode kan bedragen volgens een analyse van het Carnegie Mellon University's CyLab Sustainable Computing Consortium.[52] Het aantal varieert trouwens van programma tot programma, afhankelijk van de complexiteit van de software en de middelen die aan de ontwikkeling besteed werden. Zo had de Linux-kernel 0,17 bugs per 1000 lijnen code en de software van de Space Shuttle 0 (nul!). De impact van een bug naar de gebruiker toe kan natuurlijk sterk verschillen afhankelijk van het programma zelf en de aard van de bug; de meeste bugs worden niet als storend ervaren of treden maar in zeldzame omstandigheden op. Neem als voorbeeld een programma dat de vierkantswortel van een getal berekent: fouten kunnen zijn dat het programma numeriek een foutief antwoord geeft (bv. 3 als antwoord op de vierkantswortel van - 9), dat het blijft rekenen zonder te stoppen i.p.v. een antwoord te geven bij een niet-reguliere

[51] Meestal verkrijgt men de broncode door in de webpagina op 'Ctrl+U' te drukken of door op de rechtermuisknop en 'bron weergeven' te klikken.
[52] Zie: www.cylab.cmu.edu.

invoer (bv. bij de wortelberekening van een negatief getal of van iets dat geen getal is zoals bv. abc), dat het programma crasht bij bepaalde invoer (bv. bij het ingeven van iets dat geen getal is), enzovoort, maar meestal geeft het gewoon het goede antwoord als men een regulier positief getal invoert.

Bugs traceren en verwijderen uit softwarecode wordt debuggen genoemd. Het aantal bugs in een gegeven programma vermindert natuurlijk ook naarmate nieuwe updates van de software met foutverbeteringen (*bug fixes*) (en jammer genoeg soms ook met nieuwe bugs) worden uitgebracht. Dat verklaart meteen waarom – naast nieuwe versies met nieuwe functionaliteit – de programmeurs van een programma typisch meerdere updates van een softwarepakket uitbrengen. Bij openbronsoftware is dit debuggen zelfs een proces waaraan gans de softwaregemeenschap kan bijdragen.[51]

Vanuit juridisch standpunt kunnen we **volgende soorten gebreken in software onderscheiden** (Rinzema, 2012):

- *technische gebreken*: dit slaat op het niet of niet goed functioneren van software zonder dat aan het gebrek een gebruikershandeling ten grondslag ligt. Een voorbeeld daarvan is het vastlopen van de computer door een softwarefout of door rekenfouten.

- *functionele gebreken*: de software functioneert technisch wel, maar doet functioneel niet wat de gebruiker ervan verwachtte. De uitkomsten van het systeem zijn voor de gebruiker bijvoorbeeld niet bruikbaar (genoeg), bijvoorbeeld ze voldoen niet aan wettelijke vereisten.

- *belevingsgebreken:* de software functioneert technisch en functioneel goed, maar de gebruiker had er andere verwachtingen van. Deze belevingsgebreken zijn soms terug te voeren op interpretatieverschillen ten aanzien van de overeengekomen specificaties. Een afnemer kan de software bijvoorbeeld niet gebruiksvriendelijk genoeg vinden terwijl de softwareleverancier dat wel vindt. De gebrekkige beleving kan ook de interpretatie van het afgesloten ontwikkelingscontract zelf betreffen, bijvoorbeeld als de leverancier de werkende software niet verder ontwikkelt.

4.3. OPENBRONSOFTWARE

Software wordt ofwel aangeboden als propriëtaire software ofwel in open bron (*open source*), afhankelijk van het feit of de gebruiker al dan niet toegang krijgt tot de broncode van de software dan wel alleen de uitvoeringscode krijgt (zie ook (Truyens, 2011)).

- De meeste commerciële software is **propriëtaire software** die eigendom is van de leverancier (bv. het Microsoft Office-pakket). De gebruiker koopt maar een gelimiteerde gebruikslicentie, krijgt daarbij geen toegang tot de broncode en kan de software niet wijzigen noch wijzigingen verspreiden. Het pakket zelf is met andere woorden onaantastbaar voor de gebruiker; de leverancier is de enige die inzage heeft in de software en die wijzigingen kan en mag aanbrengen. Bugs mogen door gebruikers worden gerapporteerd, maar moeten door de leverancier worden opgelost in een update of latere versie van de software.

- Bij **openbronsoftware** (of *opensourcesoftware*) wordt ook de broncode (*source code*) van de software vrijgegeven en krijgt de gebruiker vaak de licentie om naast gebruiker ook ontwikkelaar te zijn.[53] Voorbeelden zijn de Apache webserver, de Mozilla Firefox browser of de Mozilla Thunderbird e-mailcliënt. De beschikbaarheid van de broncode geeft gebruikers de mogelijkheid om de software te bestuderen, aan te passen, te verbeteren, te verspreiden of te verkopen. De ontwikkeling van openbronsoftware komt veelal tot stand op publieke en gemeenschappelijke wijze, door samenwerking binnen een gemeenschap (community) van zowel individuele programmeurs als overheden en bedrijven. Openbronsoftware voorkomt *vendor lock-in*, waardoor men te afhankelijk zou worden van één enkele softwarefabrikant. De meeste gebruikers/ontwikkelaars proberen hun wijzigingen in het originele softwarepakket te krijgen door het bij de community aan te bieden; het onderhoud van de wijzigingen ligt in dat geval bij het originele ontwikkelteam. Als dat niet kan, dan kan de gebruiker altijd een *fork* (een eigen versie) van de software maken en dat pakket onder een gewijzigde naam publiceren en verspreiden, zelfs betalend, als dat plaatsvindt en toegelaten is binnen de voorwaarden van de verkregen licentie.

[53] Zie voor meer informatie: https://en.wikipedia.org/wiki/Open_source.

Typisch worden er bij openbronsoftware **twee licentiemodellen** gevolgd.

- → Bij de *GPL-licentie (GPL = GNU General Public License)* moet de software altijd open bron blijven, wat betekent dat ook de gewijzigde broncode altijd ter beschikking moet gesteld worden. M.a.w. bij het GPL-licentiemodel heeft de gebruiker een onbegrensd gebruiksrecht op de verdeelde broncode voor alle toepassingen, inclusief het recht om de software aan te passen en (eventueel betalend) te verspreiden, zij het onder dezelfde GPL-licentievoorwaarden (d.w.z. mét terbeschikkingstelling van de broncode en mét dezelfde GPL-rechten voor de gebruikers).

- → De *BSD-licentie (BSD = Berkeley Software Distribution)* laat wel toe om zelf propriëtaire afgeleide software van de oorspronkelijke openbronsoftware te maken en te verdelen, als de oorspronkelijke licentie maar vermeld wordt.

Leveranciers van openbronsoftware (bv. Red Hat) hebben vanzelfsprekend diepgaande kennis van de pakketten die ze ondersteunen; ze focussen zich typisch op het verlenen van extra diensten tegen betaling zoals het ondersteunen van of het schrijven van specifieke extra modules voor klanten. Dankzij het openbronmodel heeft een klant dus meer keuzevrijheid in leverancier. Ook zijn er via de community meer personen betrokken bij de softwareontwikkeling waardoor bugs sneller gevonden worden en de softwarekwaliteit hoger kan zijn. Door participatie aan discussies in de community hebben zowel gebruikers als ontwikkelaars invloed op toekomstige wijzigingen aan de software.

We dienen op te merken dat openbronsoftware niet altijd hetzelfde is als gratis software! Vele maar niet alle openbronsoftware is gratis. In 1985 werd de Free Software Foundation opgericht om een publiek besturingssysteem voor computers te ontwikkelen. Oorspronkelijk was dat het GNU project,[54] dat later leidde tot opensource BSD Unix en GNU/Linux. De vrijgave van de broncode van de Netscape Communicator webbrowser in 1998 leidde tot de oprichting van het Open Source Initiative en was het startpunt voor vele bedrijven om openbronsoftware te accepteren en aan te bieden. Sinds de vroege jaren '00 publiceerden een aantal bedrijven een deel van hun broncode terwijl ze de belangrijkste delen voor zich hielden. Dat leidde tot de nu wereldwijd

[54] GNU staat op recursieve wijze voor 'GNU's not Unix' en was een softwareproject om een volledig vrij besturingssysteem voor computers te ontwikkelen. Het heeft aanleiding gegeven tot het hier besproken GPL-licentiemodel.

gebruikte termen *Free Open Source Software* en *Commercial Open Source Software* om het onderscheid te maken tussen helemaal open en halfopen vormen van openbronsoftware. Ook de politiek van overheden zoals de Europese Unie om open toegang *(open access)* te eisen tot resultaten van onderzoek dat met publieke gelden gefinancierd wordt, bevordert dat veel softwareresultaten open bron worden en bijvoorbeeld via GitHub downloadbaar zijn.

BIBLIOGRAFIE

WETENSCHAPPELIJKE LITERATUUR

Harari, 2024: Y. N. Harari, *A brief history of information networks from the stone age to AI*, Murfreesboro: Diversified Publishing Co, 2024.

Kurzweil, 2005: R. Kurzweil, *The singularity is near: when humans transcend biology*, Viking, 2005.

Moore, 1965: G. Moore, 'Cramming more components onto integrated circuits', *Electronics Magazine,* vol. 38, No. 8, April 19, 1965.

Negroponte, 1995: N. Negroponte, *Being digital*, New York: Alfred A. Knopf, 1995.

O'Neil, 2016: C. O'Neil, *Weapons of math destruction: how big data increases inequality and threatens democracy*, New York: Crown Publishers, 2016.

Rinzema, 2012: W. F. R. Rinzema, 'Kwaliteit en software: een goede zaak', *Computerrecht*, vol. 2, pp. 96-108, 2012.

Truyens, 2011: M. Truyens, 'Open source software: een stand van zaken', *Computerrecht*, vol. 4, pp. 193-208, 2011.

Vinge, 1993: V. Vinge, 'The coming technological singularity', VISION-21 Symposium, 1993.

von Neumann, 1945: J. von Neumann, 'First draft of a report on the EDVAC', original report from 1945, reproduced in the IEEE Annals of the History of Computing, vol. 15, No. 4, pp. 27-75, 1993.

ANDERE BRONNEN

Brandon, 2018: R. Brandon (2018, 23 mei), 'Amazon needs to come clean about racial bias in its algorithms', *the Verge*. Beschikbaar: https://www.theverge.com/2018/5/23/17384632/amazon-rekognition-facial-recognition-racial-bias-audit-data.

Hunt, 2019: E. Hunt (2019, 23 januari), 'Faking it: how selfie dysmorphia is driving people to seek surgery', *the Guardian*. Beschikbaar: https://www.theguardian.com/lifeandstyle/2019/jan/23/faking-it-how-selfie-dysmorphia-is-driving-people-to-seek-surgery.

Simonite, 2018: T. Simonite (2018, 29 mei), 'How Coders Are Fighting Bias in Facial Recognition Software', *Wired*. Beschikbaar: https://www.wired.com/story/how-coders-are-fighting-bias-in-facial-recognition-software.

BRONNEN VAN FIGUREN

Figuur 1.1: door Onbekend - U.S. Army Photo, Publiek domein, https://commons.wikimedia.org/w/index.php?curid=55124Wikipedia.

Figuur 1.2: (links) CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=639279],

(midden) door Rolf Schmidt (RolfS) - Eigen werk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=404127],

(rechts) door User: Husky - eigen werk, CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=1721788.

Figuur 1.3: door Kapooht - eigen werk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=25789639.

Figuur 1.4: Max Roser, Hannah Ritchie - https://ourworldindata.org/uploads/2020/11/Transistor-Count-over-time.png, CC BY 4.0, https://commons.wikimedia.org/w/index.php?curid=98219918.

Figuur 1.5: By Courtesy of Ray Kurzweil and Kurzweil Technologies, Inc. - en:PPTExponentialGrowthof_Computing.jpg, CC BY 1.0, https://commons.wikimedia.org/w/index.php?curid=3324354Ray Kurzweil

(from R. Kurzweil, 'The age of spiritual machines', Viking Press, 1999).

Figuur 1.6: Wikipedia.

Figuur 1.7: eigen figuur.

Tabel 1.1 : eigen tabel.