Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2025, gehouden op donderdag 27 maart 2025 jl. en georganiseerd door Hogeschool Windesheim). Bij elkaar zo'n 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op 18 maart 2027 in Arnhem en wordt georganiseerd door HAN University of Applied Sciences.

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Op welke manier denkt een informaticus?

Hoe leren studenten zo te denken?

Inleiding

Op het congres NIOC 2015 zijn in de combisessie “IT-architectuur” drie onderwerpen de revue gepasseerd. Leo Pruijt lichtte in zijn presentatie met titel “Praktisch Software Architectuur Onderwijs” zijn onderzoek toe aan de hand van de tool HUSACCT. Met deze uitgekiende tool leren studenten omgaan met de samenhang van modulen in een groot informatiesysteem.
Veranderingen in de echte wereld en in veelgebruikte technologie leiden tot vragen over de doeltreffendheid van de gebruikte enterprise architectuur. Over deze veranderingen sprak Henk Plessius en stelde de vraag “Doen architecten de goede dingen?”. Hun bijdragen vindt u elders in deze bundel.
Deze beide onderwerpen hebben gemeen dat er een manier van denken aan te pas komt die typisch is voor wat van informatici verwacht wordt. Deze denkwijze onderscheidt zich doordat ze sterk gebruik maakt van het abstraherend vermogen. In dit artikel gaan we dit denken toelichten en komen vervolgens bij de vraag hoe studenten leren te abstraheren.

Belangrijke vragen in het werk van informatici

Uit de “*Great principles of Computing*” (Denning, 2008) is een aantal karakteristieke problemen af te leiden die veel voorkomen in het werk van een informaticus.

- Hoe kom je tot een design?
- Hoe kom je tot een algoritme?
- Hoe structureer je data uit verschillende typen bronnen (sensing, interactieve input, informatiesystemen)?
- Hoe leid je nieuwe kennis af uit data?
- Hoe structureer je de communicatie tussen systemen?
- Hoe evalueer je het gedrag van systemen?

Uitgangspunt is de triviale bewering: *“Een informaticus is primair een producent van ICT en niet primair een consument”*. Bij het ontwerpen van ICT-systemen is naast kennis van het toepassingsgebied inzicht in de principes van de informatica vereist. Een essentiële succesfactor ligt in het vermogen tot abstractie (Roberts, 2009, Bennedsen, 2008 en Kramer, 2007) en daarmee het vermogen complexiteit te beheersen in concrete situaties. Bij het oplossen van problemen door er software voor te schrijven gaat het om het doorgronden van het probleem zodat er specificaties voor kunnen worden opgesteld. Specificaties vormen een abstractie van het probleem (Lamport, 2015). Concrete kennis van computertechnologie, programmeertalen of andere vormen van ICT is niet van het grootste belang, maar het gaat juist om inzicht in de onderliggende principes. Het gaat evenmin om de vaardigheid ICT te kunnen gebruiken, maar om begrip van de aard van de mogelijkheden en beperkingen die voortvloeien uit enerzijds het onderhavige probleem als uit de aard van computing.

Wat is de kern van abstractie?

Een blik in de ontwikkeling van de informatica laat zien dat abstractie een wezenlijke rol gespeeld heeft. Bij het ontwerpen van onder meer programmeertalen, databasesystemen, het internet human computer interfaces wordt de onderliggende technologie bruikbaar gemaakt door te abstraheren van technische

details. De gebruiker wordt afgeschermd van grote hoeveelheden details die het zicht op geheel belemmeren. De kern van abstractie ligt in de afweging welke details weggelaten kunnen worden en welke juist zo karakteristiek zijn dat ze bijdragen aan het begrijpen van het probleem en daarom van belang blijven worden.(Kramer, 2007.) Die afweging dringt zich voortdurend op bij het beantwoorden van vragen zoals hierboven gesteld zijn en zeker bij het vertalen van een probleem in de echte wereld naar een oplossing in de vorm van software. Dit kan betekenen dat bij sommige kwesties details belangrijk worden die er bij andere niet toe doen. In dit verband is de waarschuwing van Brooks interessant*: Hence, descriptions of a software entity that abstract away its complexity often abstract away its essence*. (Brooks, 1987). Niet elke abstractie is zinvol. De essentie moet behouden blijven. Dit geldt zeker ten aanzien van security waarbij problemen ontstaan als bepaalde details die vanuit een functionele view minder belangrijk zijn, verwaarloosd worden.

Naast het onderkennen van de details die terzake doen is een tweede activiteit van een informaticus het omzetten van specificaties in concrete computersystemen, software en informatiesystemen. Dit vertalen van vereisten vraagt om denken in lagen en concepten - ook vormen van abstractie!- en is daarnaast aan strakke regels gebonden waarbij het een beroep doet op het zorgvuldig kunnen hanteren van formele werkwijzen. Vaak wordt aan dit laatste, formele aspect meer aandacht gegeven dan aan het denken en ontwerpen. Daardoor is het risico groot dat studenten in de valkuil lopen regeltjes en trucjes toe te passen zonder analyse en inzicht. Er wordt bij voorbeeld een keus gemaakt voor een design pattern of een programmeertaal omdat die nu eenmaal bekend zijn, zonder de afweging of deze geschikt zijn in de onderhavige situatie. Het resultaat is dan een systeem zonder elegantie dat mogelijk niet doet wat ervan verlangd wordt en bovendien slecht te onderhouden is.

Hoe leer je abstraheren?

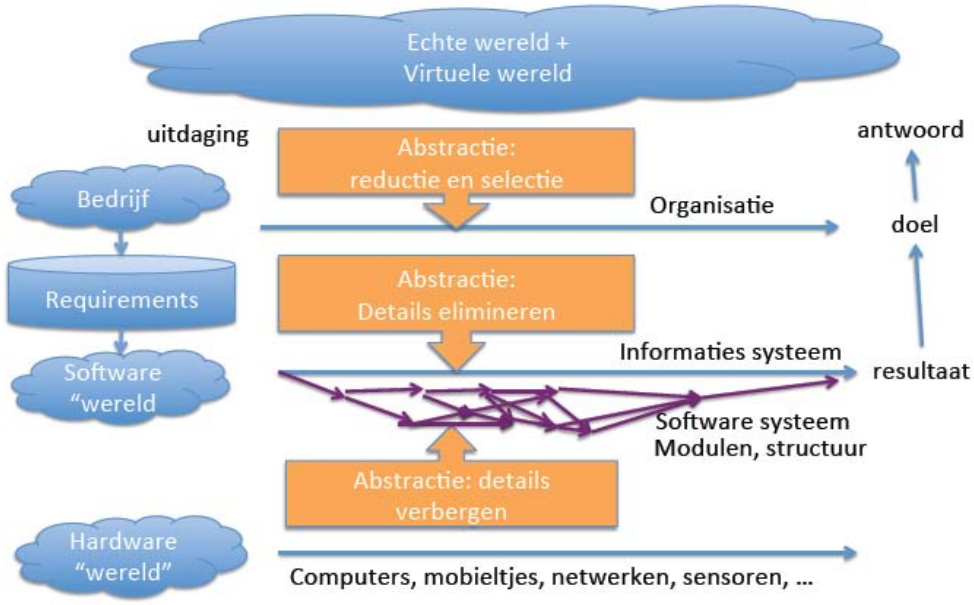
De vraag is hoe het onderwijs ingericht en uitgevoerd kan worden zodat leerlingen en studenten betere informatici voor de beroepspraktijk worden. Het gaat dus niet alleen om kennis van enkele methoden (patronen) van ontwerpen, informatieanalyse, programmeren, maar zelf creatief en verantwoord te komen tot keuzen van hulpmiddelen en (deel)oplossingen om systemen te creëren die ertoe doen. In het verlengde hiervan dringt de vraag zich op hoe komen studenten en leerlingen tot deze manier va denken.

Een klassiek antwoord is dat wiskunde (Borovik, 2011) daarvoor nodig is. Anderen benadrukken juist het belang van taalbeheersing, weer anderen wijzen op het denken in modellen zoals in de natuurkunde. De uitspraak van Keith Devlin is zinvol om als uitgangspunt te nemen: “Once we have learned how to reason precisely about one set of abstracties, it take relatively little extra effort to reason about any others” (Devlin, 2003).

Desondanks is het de moeite waard na te gaan of er binnen het domein van de informatica zelf niet onderwerpen zijn die het abstract denken bevorderen. Daar zijn in de literatuur pogingen te vinden. Armoni en Gal_Ezer gebruiken oefeningen met reguliere expressies (Armoni, 2006). Tegenwoordig is er veel aandacht voor Computional Thinking (Wing, 2006) en worden cursussen gegeven waarin eenvoudig programmeren in Scratch als doel heeft abstract denken te combineren met het opstellen van algoritmen. Een voorbeeld daarvan is de cursus CS10 “The Beauty and Joy of Computing” op Berkeley (http://inst.eecs.berkeley.edu/~cs10/fa14). Ook het leren gebruiken van UML diagrammen met de bijbehorende user stories zou kunnen helpen. Bovendien leert de ervaring dat abstractie een vaardigheid is die alleen door oefening geleerd wordt (Lamport, 2015). Laat die ervaring dan ook bij voorkeur gevarieerd zijn, zodat niet steeds vanuit hetzelfde gezichtspunt problemen kunnen worden opgelost.

Abstractie in de combisessie IT-architectuur

In de combisessie over IT-architectuur op NIOC2015 hebben we abstractie gezien in verschillende vormen. Het volgende schema geeft globaal de samenhang tussen de drie deelsessies weer.



Het is verrassend te zien hoeveel samenhang in de manier van denken te zien is in de aanpak van uiteenlopende problemen. Die samenhang ontstaat door te onderkennen dat abstraheren voortdurend vereist wordt.

Referenties

Armoni, M., Al-Ezer, J. (2006). Reduction – an Abstract Thinking Pattern: The Case of the Computational Models Course, **ACM SIGCSE'06,** March 1–5

Bennedsen, J., Capersen, M.E. (2008). Abstraction Ability as an Indicator of Success for Learning Computing Science?, **ACM SICER'08**, September 6–7, 2008

Borovik, A.V. (2011). The strange fate of abstract thinking, www.computer9.net/t/the-strange-fate-of-abstract-thinking-w7903.html, geraadpleegd op 9 december 2014

Brooks, F. P. (1987). No Silver Bullet: Essence and Accidents of Software Engineering, **Computer**, Vol. 20, No. 4, 10-19.

Denning, P.J. (2008). Great Principles of Computing, geraadpleegd op 11 december 2014 op http://denninginstitute.com/pjd/PUBS/ENC/gp08.pdf

Devlin, K . (2003). Why universities require computer science students to take math , **Communications of ACM,** 46 (9), 37-39

Kramer, J. (2007). Is abstraction the key to computing?, **Communications of the ACM**, 50(4), 37-42

Lamport, L. (2015). Who Builds a House without Drawing Blueprints?, **Communications of the ACM**, 58(4), 38 - 41

Roberts, P. (2009). Abstract Thinking: a predicator of modelling ability, **Proceedings Educators Symposium, MODELS 2009**, ACM/IEEEE

Wing, 2006). Computational Thinking, **Communications of ACM,** 49 (3)