Feedback voor deelopdracht 3 was dat ons artikel niet een factchecker was, de stelling niet goed was en dat het opnieuw geschreven moest worden en dat de stelling aangepast wederom diende aangepast te worden. De stelling was namelijk tussen deelopdracht 1 en deelopdracht 2 in het begin aangepast, omdat deze te moeilijk geformuleerd was en er niet of nauwelijks over schreven kon worden.

Hieronder is dus ook deelopdracht 3, het opnieuw geschreven artikel in concept vorm te vinden, als bewijsvoering.

Inleiding

Voor het maken van programma's wordt er gebruik gemaakt van programmeertalen. Door het gebruik van een programmeertaal is het mogelijk voor een programmeur om een computer te voorzien van instructies en zo de gigantische rekenkracht van het apparaat te benutten. Er zijn door de jaren heen veel programmeertalen ontwikkelt. Sommige revolutionair en vernieuwend, andere een uitbreiding op reeds bestaande talen. Veel van deze programmeertalen delen dezelfde kenmerken en kunnen worden geclassificeerd onder een *'paradigma'*.

Het is daarom van essentieel belang dat een keuze voor een paradigma afhankelijk is van een correcte probleem- of doelstelling. Zo wordt er vanuit de academische en wetenschappelijke hoek al jaren aangezet tot een transitie naar het functionele paradigma, maar worden functionele programmeertalen, zoals Haskell en Lisp vaak als een onconventionele, onwerkbare vorm van software ontwikkeling gezien. Echter wordt er momenteel met de komst van '*big data'* en *'cloud computing'* een nieuwe efficiënte manier van dataverwerking toegepast die veelal gebruik maakt van parallel, gelijktijdig en gedistribueerd programmeren.

Deze oplossingen schikken zich vaak voor een functionele aanpak van het probleem, maar de achterliggende infrastructuur is vaak imperatief en objectgeoriënteerd, waardoor een combinatie veelal uitgesloten is, maar wel wenselijk.

Dit rapport bekijkt samen met de lezer de voor en nadelen bij de stelling '***Het functionele paradigma is de toekomst.'*** Dit gebeurt op basis van een literatuuronderzoek naar de relaties en verschillen van de heersende programmeerparadigma's in de industrie. Het onderzoek beperkt zich tot het imperatieve en functionele paradigma. Om het functionele paradigma en zijn plaats binnen de informatica beter te kunnen begrijpen is er binnen dit onderzoek van een minimale kennisgeving van de heersende programmeerparadigma's in de industrie uitgegaan. Ook is kennis van programmeren wenselijk, maar kan het onderzoek ook prima zonder deze kennis worden gevolgd.

Middenstuk

In dit middenstuk zal er een korte uitleg worden gegeven wat de imperatieve en functionele programmeerparadigma's inhouden. Opvolgend zal er worden gekeken naar twee kanten van deze stelling, namelijk de voor en tegenargumentatie.

Om de plaats van het functionele programmeerparadigma beter te kunnen begrijpen, zal er eerst gekeken moeten worden naar de andere dominante paradigma's in de industrie. De paradigma's die kort besproken zullen worden zijn: het imperatieve en functionele programmeerparadigma.

Imperatief programmeren:

Imperatief programmeren houdt in dat een programma een structuur beschrijft volgens het Turing-model. Dit programma wordt door de computer ingelezen in het geheugen om het in kleine delen *'sequentieel'* uit te voeren. De tussentijdse berekeningen worden in een imperatieve taal als data opgeslagen in het geheugen. Deze data kan vervolgens worden opgehaald en *'gemuteerd'* door elk volgende instructie dit toegang heeft tot deze data, totdat er een antwoord wordt gepresenteerd. Imperatieve programmeertalen zijn door deze eigenschap om de huidige staat van het programma aan te passen zeer geschikt voor het programmeren van computer hardware. Zo zijn CPU's imperatieve executie machines en compilers vertalen direct naar deze taal. Dit is de reden dat imperatief programmeren hedendaags de gangbare manier van programmeren is.

Functioneel programmeren:

Functioneel programmeren is een paradigma dat niet nieuw is, maar al sinds de jaren 40 bestaat. Het is een paradigma wat is ontstaan uit het *'lambda calculus'*, wat Turingvolledig is. Kenmerkend voor functioneel programmeren is het programmeren met functies, het vermijden van globale staat , de niet muteerbare data structuren en een executie model waarin de evaluatie volgorde niet uitmaakt.

Algemene voordelen functionele programmeren

Nu er kort uitleg is gegeven over imperatieve en functionele talen kunnen we kijken naar de verschillen, in het bijzonder de voor en nadelen. Om de verschillen echter goed te begrijpen is het handig om te weten dat op het moment van schrijven van dit rapport het dominante paradigma in de industrie de imperatieve manier van programmeren is. Het is echter pas sinds kort dat er hernieuwde interesse is ontstaan in de functionele talen vanwege de expressiviteit en het vermogen van deze talen om foutloos gelijktijdig programma's uit te voeren. Deze eigenschappen sluiten heel goed aan bij de belangen van veel bedrijven die met de komst van het internet, *'cloud computing'* en *'big data'* over data beschikken die zo omvangrijk is dat de traditionele data verwerkingssoftware niet toereikend genoeg is om hiermee om te gaan. De oplossing: het verdelen van de data over verschillende multi-core processoren, zodat er betere prestaties kunnen worden behaald en de verwerking sneller verloopt. Dit heet in de informatica gedistribueerd programmeren.

Gedistribueerd programmeren is een complex onderwerp in de informatica niet zonder enige risico's. Doordat er meerdere multi-core processoren aangesproken worden, wat tevens gelijktijdig kan gebeuren, moet er goed nagedacht worden over de evaluatie volgorde. Doordat de aangesproken processoren niet dezelfde *'throughput'* hebben zijn uitspraken over ruimte en tijd complexiteit moeilijk voor een programmeur. Zeker als we er vanuit kunnen gaan dat het meest gebruikte paradigma in de industrie de imperatieve stijl van programmeren is. Het imperatief programmeren is inherent sequentieel en data is muteerbaar wat het correct uitvoeren van een programma nog meer bemoeilijkt door eventuele *'data races'*. Om data races te voorkomen zijn er in vele imperatieve programmeertalen *'locks'* of *'mutexes'* geïntroduceerd, met als doel het correct afhandelen van de executievolgorde van een imperatief programma. Echter zijn de geïntroduceerde oplossingen niet zonder bijwerkingen, denk aan de *'deadlock'*.

Functionele programmeertalen hebben echter veel van deze problemen niet. Zo zijn functionele talen niet afhankelijk van de evaluatie volgorde en zijn er geen *'locks'* of *'mutexes'*, wat ze uitsluit voor *'deadlocks'*. Dit maakt het mogelijk om verschillende stukken van een *'expressie'* parallel te evalueren, wat een pre is voor gedistribueerd programmeren. Tevens is de data in de functionele programmeertalen niet muteerbaar, wat inhoudt dat er geen bijwerkingen kunnen ontstaan, waardoor *'data races'* niet tot nauwelijks voor kunnen komen in deze talen.

De functionele programmeertalen zijn ook erg schaalbaar, zo zijn bepaalde algoritmes in functionele programmeertaal vaak minder regels code dan in een imperatieve taal, doordat functionele talen vaak een hogere abstractie hanteren, denk hierbij aan het gebruik van *'hogere graads functies'*. Dit zorgt voor een duidelijker en beter herbruikbaar systeem, waar elk stuk code een eigen, gescheiden verantwoordelijkheid heeft, dit heet modulariteit.

Algemene nadelen functioneel programmeren

Functioneel programmeren kent zo echter ook zijn programmeertalen zijn er nadelen wat betreft het gebruik van functioneel programmeren. Zo zijn vrijwel vaak functionele programmeertalen langzamer ten opzichte van imperatieve programmeertalen, aangezien deze talen focussen op abstractie, waardoor de programmeur minder mogelijkheden heeft om direct het onderliggende geheugengebruik te controleren.

Ook maken de meeste functionele programmeertalen gebruik van static type checking. Wanneer een verkeerd type variabele wordt gebruikt, zal het programma niet werken.

Waar in andere programmeerparadigma's veel gebruik kan worden gemaakt van libraries, is dit bij functioneel programmeren niet/minder vaak het geval, op een paar libraries na.

Wat bij programmeren tot één van de meest belangrijke zaken behoort, is portability. Dit is niet het geval bij programma's geschreven in een functionele programmeertaal. Omdat het omzetten hiervan erg lastig is.

Ook het gebruik van programma's gebouwd met functionele programmeertalen, kunnen nadelig zijn. Omdat er allerlei extra bestanden worden bijgeleverd om het programma te kunnen gebruiken, kost dit ook meer qua werkgeheugen, wat betekent dat daar ook rekening mee moet worden gehouden, met het kiezen van functionele talen over object georiënteerd.

Bijlage beschrijvingen

Paradigma:

Collectie van concepten en denkpatronen die een aanpak voor een doel of probleem beschrijven.

Portability

Het gebruik van een programma op meerdere besturingssystemen kan worden gebruikt

Conclusie

Bronnenlijst

Barendregt, H., & Barendsen, E. (2000, maart). Introduction to Lambda Calculus [PDF]. Geraadpleegd van ftp://ftp.cs.ru.nl/pub/CompMath.Found/lambda.pdf

Cass, S. (2017, 18 juli). The 2017 top programming languages. Geraadpleegd van https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages

Dwarampudi, V. (2010). Comparative study of the Pros and Cons of Programming Languages: Java, Scala, C++, Haskell, VB .NET, AspectJ, Perl, Ruby, PHP & Scheme [PDF]. Geraadpleegd van https://arxiv.org/pdf/1008.3431.pdf

Hughes, J. (1990). Why Functional Programming Matters [PDF]. Geraadpleegd van https://www.cs.kent.ac.uk/people/staff/dat/miranda/whyfp90.pdf

Ipovaca, M. (2011). Learn You a Haskell for Great Good! [PDF]. Geraadpleegd van https://github.com/clojurians-org/haskell-ebook/raw/master/Learn%20You%20a%20Haskell%20for%20Great%20Good!%20A%20Beginner's%20Guide.pdf

Jones, S. P. (2008). Harnessing the Multicores: Nested Data Parallelism in Haskell [PDF]. Geraadpleegd van https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/fsttcs2008.pdf

Jones, S. P. (2011). The future is parallel, and the future of parallel is declarative [PDF]. Geraadpleegd van http://www.bcs.org/upload/pdf/future-is-parallel-140512.pdf

Jones, S. P. (2011, 17 december). Haskell is useless [Video]. Geraadpleegd van https://www.youtube.com/watch?v=iSmkqocn0oQ

Jones, S. P., & Harris, T. (2012, 21 november). Programming in the Age of Concurrency: Software Transactional Memory [Video]. Geraadpleegd van https://www.youtube.com/watch?v=aQXgW55f7cg

Lisper, B. (2013, juli). Functional Programming and Parallel Computing [PDF]. Geraadpleegd van http://www.idt.mdh.se/kurser/DVA201/slides/parallel.pdf

Petricek, T., & Skeet, J. (2010). Real-World Functional Programming: With Examples in F# and C# [PDF]. Geraadpleegd van http://www.gbv.de/dms/ilmenau/toc/593627342.PDF

Svensson, K., & Eliasson, T. (2017, 26 juni). A comparison of functional and object-oriented programming paradigms in JavaScript [PDF]. Geraadpleegd van http://www.diva-portal.se/smash/get/diva2:1115428/FULLTEXT02.pdf

TIOBE Index for December 2017. (z.j.). Geraadpleegd van http://www.tiobe.com/tiobe-index

Wadler, P. (1992, februari). The essence of functional programming [PDF]. Geraadpleegd van https://page.mi.fu-berlin.de/scravy/realworldhaskell/materialien/the-essence-of-functional-programming.pdf

Wadler, P. (1998, juli). Why No One Uses Functional Languages [PDF]. Geraadpleegd van http://web.vtc.edu/users/pcc09070/cis-3030/Why-Not-FP.pdf

Wadler, P. (1998). The Expression Problem [E-mail]. Geraadpleegd van http://homepages.inf.ed.ac.uk/wadler/papers/expression/expression.txt