onderzoeksverslag Functioneel Paradigma

Jelmer van Vugt (618770) – 21-10-2021

# Abstract

dd

Contents

[Abstract 1](#_Toc54356060)

[Lijst met afkortingen 3](#_Toc54356061)

[Begrippenlijst 4](#_Toc54356062)

[1. Inleiding 5](#_Toc54356063)

[2. Methodologie 6](#_Toc54356064)

[2.2 Literatuurstudie 6](#_Toc54356065)

[2.2 Prototyping 6](#_Toc54356066)

[3. Onderzoeksresultaten 7](#_Toc54356067)

[3.1 Literatuuronderzoek 7](#_Toc54356068)

[3.1.1 Wat is imperatief en declaratief programmeren? 7](#_Toc54356069)

[3.1.2 Welke termen komen veel voor binnen het functionele paradigma? 7](#_Toc54356070)

[3.1.3 Wat zijn states en hoe hebben deze invloed op het functionele paradigma? 9](#_Toc54356071)

[3.2 Labonderzoek 9](#_Toc54356072)

[3. Conclusie 10](#_Toc54356073)

[4. Discussie 11](#_Toc54356074)

[5. Literatuurlijst 12](#_Toc54356075)

[6. Bijlagen 13](#_Toc54356076)

# Lijst met afkortingen

ff

# Begrippenlijst

ff

# 1. Inleiding

In dit onderzoeksverslag – wat in opdracht is van de course Algoritmes, Programmeertalen en Paradigma’s in het derde jaar Software Development zoals aangeboden door de Hogeschool van Arnhem en Nijmegen – is de hoofdvraag ‘Wat is functioneel programmeren?’ beantwoord. Dit is gedaan door middel van verschillende deelvragen waarop in de hier op volgende leeswijzer verder op in wordt gegaan.

In het eerste hoofdstuk ‘Methodologie’ worden de verschillende onderzoeksmethoden besproken die gehanteerd zijn tijdens het beantwoorden van de deelvragen. In het daaropvolgende hoofdstuk ‘Onderzoeksresultaten’ zijn de resultaten van de volgende deelvragen beantwoord: Wat is imperatief en declaratief programmeren? Wat zijn states en hoe hebben deze invloed op het functionele paradigma? Welke termen komen veel voor binnen het functionele paradigma? Hoe schrijf je een JSON parser in de programmeertaal Haskell? Behalve de laatste deelvraag zijn deze allemaal onderzocht door middel van een literatuuronderzoek, voor de laatste is een workshop onderzoek in de vorm van een prototype volbracht. Hierna zal in hoofdstuk drie de hoofdvraag worden beantwoord met behulp van de in het vorige hoofdstuk vergaarde resultaten. Ook wordt er in hoofdstuk vier gereflecteerd op de getrokken conclusie. Tot slot zijn in de laatste twee hoofdstukken de literatuurlijst en de bijlagen te vinden.

# 2. Methodologie

In dit hoofdstuk worden de twee verschillende onderzoeksmethoden toegelicht die gebruikt zijn om de deelvragen te beantwoorden.

## 2.2 Literatuurstudie

Binnen een literatuurstudie wordt informatie vergaard uit leesbare artikelen. Deze informatie kan afkomstig zijn van websites, boeken, tijdschriften, kranten, et cetera. Uit deze gevonden artikelen kunnen selecte stukken worden gebruikt om op deze manier bij te dragen bij het beantwoorden van de deelvraag.

## 2.2 Prototyping

Binnen de prototyping onderzoeksmethode wordt een prototype ontwikkeld om een concept, design of probleem beter te begrijpen en/of de technische limieten en mogelijkheden de ontdekken.

# 3. Onderzoeksresultaten

In dit hoofdstuk zullen de resultaten van elke deelvraag worden behandeld.

## 3.1 Literatuuronderzoek

Binnen deze paragraaf zullen de deelvragen worden behandeld die door middel van een literatuuronderzoek zijn beantwoord.

### 3.1.1 Wat is imperatief en declaratief programmeren?

Voordat er verder kan worden ingegaan op wat functioneel programmeren precies inhoud is het belangrijk eerst te weten wat imperatief – en declaratief programmeren is.   
Het grootste verschil tussen de twee paradigma’s ligt in het feit dat imperatief programmeren beschrijft *hoe* een programma werkt en declaratief *wat* een programma moet doen. Om hier een beter beeld bij te illustreren pakken we voor het gemak het bouwen van een huis. Imperatieve programma’s zouden de volgende structuur hebben: bouw de fundering, voeg het skelet toe, installeer de utiliteiten, bouw de muren en werk het tot slot mooi af. Een aanroep om een huis te bouwen in een declaratief programma zou er als het volgende uitzien: Ik wil een huis met een open haard en drie verdiepingen, het maakt me niet uit hoe je dit doet. Dit wordt ook wel abstracte code genoemd.   
Talen die gebruik maken van het imperatieve paradigma worden ook wel procedurele talen genoemd. Voorbeelden hiervan zijn Java en C++. Talen die gebruik maken van het declaratieve paradigma worden ook wel functionele talen genoemd. Voorbeelden hiervan zijn Haskell en Lisp.

### 3.1.2 Welke termen komen veel voor binnen het functionele paradigma?

Onder dit kopje worden een aantal veel voorkomende termen binnen het functionele paradigma onder elkaar gezet en toegelicht.

#### 3.1.2.1 Purity

In sommige functionele programmeertalen is het mogelijk voor functies om naast de returnwaarde te berekenen en terug te geven ook overige acties te ondernemen. Deze overige acties worden ook wel *side-effects* genoemd om te benadrukken dat de returnwaarde het belangrijkste gedeelte van de functie is.

Binnen het functionele paradigma kunnen functies puur genoemd worden als ze voldoen aan de volgende twee eisen:

* De functie moet altijd dezelfde waarde teruggeven gegeven dezelfde inputwaarden.
* Het uitvoeren van de functie heeft geen bijwerkingen op de applicatie. Een voorbeeld hiervan is het bijwerken van een variabele buiten de functie zelf.

Omdat pure functies altijd dezelfde output moeten geven met dezelfde input zijn deze erg gemakkelijk te unit-testen. Ook hebben pure functies omdat deze geen interactie hebben met de buitenwereld amper impact op concurrency, wat erg voordelig is.

##### Lazy evaluation

Omdat pure functies gegeven dezelfde argumenten altijd dezelfde uitkomst hebben kunnen de uitkomsten van bepaalde functies gegeven dezelfde argumenten worden hergebruikt. Hierdoor worden onnodige computaties voorkomen.

##### Immutable data

Pure functionele programma’s maken meestal gebruik van *immutable data.* In plaats van het wijzigen van bestaande waarden worden nieuwe kopieën gemaakt waardoor de oude waarde wordt behouden. Omdat de originele waarden van het systeem niet gewijzigd kunnen worden kunnen deze gedeeld en hergebruikt worden. Wat geheugen scheelt.

##### Monads

Tijdens het berekenen van de waarde die een functie moet teruggeven kunnen er allerlei dingen misgaan. Denk hierbij aan: de waarde bestaat niet, er zijn meerdere instanties van de gezochte waarde, er komt een I/O operatie bij het krijgen van de waarde aan te pas, de berekening produceert een error, et cetera.

Een monad fungeert als een container die door abstractie net doet alsof deze eigenaardigheden niet bestaan. Hierdoor kan de focus worden gelegd op wat er met de waarden gedaan moet worden. Ook brengt de monad een set operaties met zich mee die uitgevoerd kunnen worden op deze container. Hierbij kan gedacht worden aan: map, return, join en bind.

#### 3.1.2.2 Higher order functions

Higher-order functions (HOFs) zijn functies die een functie gebruiken als een argument of een functie gebruikt als return waarden. Een HOF kan dit ook beide doen. Een voorbeeld hiervan is een functie die zowel een lijst als een functie krijgt als argumenten. Vervolgens voert de functie op elk item binnen de lijst de meegegeven functie uit en geeft de nieuwe lijst terug.

Een van de voordelen die HOFs met zich meebrengen is het verminderen van repetitie binnen de code. Neem als voorbeeld een for lus. Deze bestaan altijd, exclusief de implementatie binnen de lus, uit dezelfde structuur. Het herhalen van deze lussen kan opgevangen worden met een HOF.

#### 3.1.2.2 Currying

Currying is het proces waarin één functie die meerdere argumenten ontvangt wordt opgebroken in meerdere individuele functies die elk maar een enkel argument ontvangen en in serie staan. Elke functie berekend een waarde gebaseerd op zijn argument en geeft deze terug aan de functie boven hem in de hiërarchie.

Door één grote functie op te splitsen in verschillende sub-functies wordt de code makkelijker om te refactoren. Ook maakt dit de code makkelijker te lezen én te begrijpen voor de ontwikkelaar(s).

#### 3.1.2.3 Partial application

snap het verschil met currying nog niet zo goed

### 3.1.3 Welke rol spelen states binnen het functionele paradigma?

Jj

## 3.2 Labonderzoek

Binnen deze paragraaf zullen de resultaten van alle deelvragen worden behandeld die met een labonderzoek zijn verkregen.

# 4. Conclusie

ff

# 5. Discussie

ff

# 6. Literatuurlijst

Golliwitzer, Z. (2020) *Imperative vs. Declarative Programming (procedural, functional, and OOP).* Geraadpleegd op 18 oktober 2020, via:<https://medium.com/@zach.gollwitzer/imperative-vs-declarative-programming-procedural-functional-and-oop-b03a53ba745c>.

Ram, M. (2019) *Functional Programming in simple terms.* Geraarpleegd op 18 oktober 2020, via: <https://medium.com/angular-in-depth/functional-programming-in-simple-terms-abcef30a2ad1>.

Brassuer, A. (2014) *Functional Programming: Pure Functions.* Geraadpleegd op: 18 oktober 2020, via: <https://www.sitepoint.com/functional-programming-pure-functions/#:~:text=A%20pure%20function%20is%20a,always%20return%20the%20same%20result>.

Khan, A. (2015) *What is functional programming?* Geraadpleegd op 23 oktober 2020, via: <https://wiki.haskell.org/Functional_programming#What_is_functional_programming.3F>.

Haskell Wiki. (2020) *Monads.* Geraadpleegd op 23 oktober 2020, via: <https://wiki.haskell.org/Monad>.

Haskell Wiki.(2020) *Currying.* Geraadpleegd op 23 oktober 2020, via: <https://wiki.haskell.org/Currying>.

Gupta, D. (2018) *Javascript – Currying VS Partial Application.* Geraadpleegd op 23 oktober 2020, via: <https://towardsdatascience.com/javascript-currying-vs-partial-application-4db5b2442be8>.

# 7. Bijlagen

fff