The Framework Productivitity Measurement Method

Meten van de productiviteitwinst bij het gebruik van een webframework

Masterscriptie Paul Bakker
Master Software Engineering
Universiteit van Amsterdam

Afstudeerdocent	:	Jurgen Vinju
Stagebegeleider	:	Ronald Verduin
Opdrachtgever	:	Info Support BV
Publicatiestatus	:	openbaar
Versie	:	1.4
Datum	:	11 augustus 2006

1 Voorwoord

Deze scriptie is geschreven tijdens mijn afstudeerperiode van de Master Software Engineering aan de Universiteit van Amsterdam.
De opdracht is tot stand gekomen door mijn eigen interesse in het ontwikkelen van webapplicaties, en de visie van Info Support BV hier op.
In het bijzonder wil ik Ronald Verduin en Jurgen Vinju bedanken voor hun begeleiding.

Paul Bakker

2 Inhoud

3 Samenvatting

Info Support B.V. moet bij klanten advies kunnen geven over het al dan niet gebruik van het JSF framework. JSF is onderdeel van de JEE 5 specificatie en is een framework voor component gebaseerde webontwikkeling. Een van de belangrijkste vraagstukken bij het geven van zulk advies, is welke invloed JSF heeft op de productiviteit van een ontwikkelaar in verhouding tot andere webframeworks.

Om antwoord te vinden op het productiviteitsvraagstuk, is er een methode ontwikkeld waarmee de productiviteitswinst van een webframework kan worden gemeten. De ontwikkelde methode, de Framework Productivity Measurement Method, is zo ontwikkeld dat de resultaten van FPMM zo veel mogelijk onafhankelijk zijn van het niveau van de ontwikkelaar die de methode gebruikt. Om dit mogelijk te maken wordt een framework met FPMM geëvalueerd op basis van een vergelijking met een referentie framework.

Een ander belangrijk punt van FPMM is dat de evaluatie van een framework niet is gebaseerd op een architecturale analyse, maar op een praktijk test. Uitspraken doen over de productiviteitswinst die een framework biedt op basis van de architectuur van een framework is niet realistisch. Een goed ontworpen architectuur is nog geen garantie voor een hoge productiviteit.

De praktijktest wordt gedaan door een gespecificeerde testapplicatie te implementeren, zowel met het te evalueren framework, als met het referentie framework. Hierbij wordt de gebruikte tijd vastgelegd. Na de implementatie met beide frameworks wordt er geanalyseerd wat het verschil is tussen de meetwaarden van beide frameworks. Op basis hiervan zal blijken dat er met framework A, N keer productiever kan worden gewerkt dan met framework B.

Bij het implementeren van de testapplicatie, wordt tijd besteed aan debuggen en research niet meegeteld als ontwikkeltijd. Debuggen is een niet te vermijden onderdeel uit het software constructie proces, maar is teveel afhankelijk van de ervaring van een ontwikkelaar, en de ervaring met een specifieke techniek. Als debug tijd zou worden meegeteld als ontwikkeltijd, is het niet mogelijk uitspraken te doen over de productiviteit, onafhankelijk van de ervaring van de ontwikkelaar die de evaluatie doet. Dit zou de resultaten van de evaluatie veel minder bruikbaar maken.

Om antwoord te kunnen geven op de vraag van Info Support B.V., is FPMM toegepast in een casestudy waarbij het JSF framework is geëvalueerd. Deze casestudy kan worden gezien als een voorbeeld van het gebruik van FPMM.

In de casestudy is het Spring WebMVC framework gebruikt als referentie framework, omdat hier veel ervaring van aanwezig is. Uit de casestudy blijkt dat de meeste onderdelen van de testapplicatie met een factor 1,5 sneller geïmplementeerd konden worden dan met Spring WebMVC. Dit is een significante winst en is zeker een overtuigend argument om JSF te adopteren in een project. Hierbij moet wel rekening worden gehouden met de iets steile leercurve van JSF die veroorzaakt wordt door het vrij complexe component model waar JSF op gebaseerd is.

Door de component gebaseerde manier van werken, wordt er enige flexibiliteit ingeleverd met betrekking tot de layout van pagina's. Dit kan in bepaalde typen applicaties nadelig zijn, bijvoorbeeld bij applicaties waar een geavanceerde layout noodzakelijk is. Per project kan er een afweging worden gemaakt of dit punt zwaarder weegt dan de productiviteitswinst die JSF biedt bij het implementeren van andere onderdelen van de applicatie.

4 Inleiding onderzoek

4.1 Onderzoeksvraag

4.1.1 Achtergrond

Info Support BV zal bij klanten advies moeten geven over het al dan niet gebruik van JSF. Een belangrijk vraagstuk hierbij is wat de productiviteitswinst is die JSF biedt ten opzichte van reeds in de praktijk bewezen frameworks zoals Struts en Spring WebMVC.

Het doel van dit onderzoek is om hier antwoord op te vinden; wat is de productiviteitswinst bij het gebruik van het JSF framework? Om hier antwoord op te vinden wordt er eerst onderzoek gedaan naar een meer algemene vraag, namelijk hoe de productiviteitwinst van een (web)framework gemeten kan worden. Hier moet een onderzoeksmethode voor ontwikkeld worden, die gebruikt kan worden om verschillende webframeworks te evalueren.

Deze methode wordt vervolgens gebruikt om het JSF framework te evalueren. Het toepassen van de onderzoeksmethode op het JSF framework kan worden gezien als een casestudy, en dient als voorbeeld voor het gebruik van de onderzoeksmethode die gepresenteerd wordt.

Als onderdeel van de casestudy is het zaak om te onderzoeken welke implementatie van JSF op dit moment de beste keuze voor gebruik is.

4.1.2 Onderzoeksvraag

Gebaseerd op het bovenstaande, is de volgende onderzoeksvraag opgesteld.
“Hoe kan de productiviteitswinst van het gebruik van een gegeven framework ten opzichte van een referentie framework worden gemeten?” De exacte definitie voor productiviteit en productiviteitswinst is gegeven in hoofdstuk 6.1.1.

De hoofdvraag is opgedeeld in een aantal deelvragen:

- Hoe kan productiviteit worden gemeten?
- Hoe kan de productiviteitswinst van een framework worden gemeten?
- Hoe kunnen verschillende webframeworks met elkaar worden vergeleken?

4.1.3 Te behalen resultaten

Als resultaat van het onderzoek naar bovengenoemde vragen wordt er een methode ontwikkeld waarmee webframeworks kunnen worden geëvalueerd op de productiviteitswinst die ze bieden. Deze methode moet bruikbaar zijn voor verschillende personen om webframeworks te evalueren.

4.1.4 Casestudy

Om antwoord te kunnen geven op de oorspronkelijke vraag van Info Support BV, wordt de gepresenteerde methode toegepast in een casestudy, waarbij het JSF framework en de verschillende implementaties hiervan wordt geëvalueerd. Op basis van deze evaluatie kan Info Support BV advies uitbrengen aan klanten over het gebruik van JSF.

4.2 Aanpak

Het onderzoek is opgedeeld in een aantal stappen. Deze stappen worden kort in deze paragraaf toegelicht:

- Literatuuronderzoek
- Beschrijven methode
- Uitvoeren casestudy
- Terugblik / Analyse methode

4.2.1 Literatuuronderzoek

Tijdens het literatuuronderzoek wordt onderzocht welk onderzoek er al gedaan is naar het meten van productiviteit, en welk onderzoek er gedaan is naar het beschrijven van een framework. Ook wordt er literatuur gelezen over onderwerpen die gerelateerd zijn aan de context van het onderzoek, namelijk het ontwikkelen van web applicaties.

4.2.2 Ontwikkelen methode

Na het literatuuronderzoek wordt de opgedane kennis gebruikt om een methode te ontwikkelen voor het meten van de productiviteitswinst dat een webframework kan bieden. Bij het beschrijven van deze methode worden gemaakte keuzes onderbouwt met behulp van gevonden literatuur. De methode is het belangrijkste resultaat uit dit onderzoek.

4.2.3 Uitvoeren casestudy

Om de ontwikkelde methode te valideren, en om antwoord te kunnen geven op de door Info Support B.V. gestelde vraag, wordt de ontwikkelde methode toegepast in een casestudy. De methode zal hierbij gebruikt worden om het JSF framework te evalueren.

4.2.4 Terugblik / Analyse methode

Na het uitvoeren van de casestudy moet het duidelijk zijn of de ontwikkelde methode in de praktijk bruikbaar is. Tijdens het onderzoek is het niet mogelijk de ontwikkelde methode te testen met een grote groep ontwikkelaars. Hierdoor kan de casestudy geen volledige garantie geven dat de methode in alle gevallen, onafhankelijk van het niveau van een ontwikkelaar, werkt.

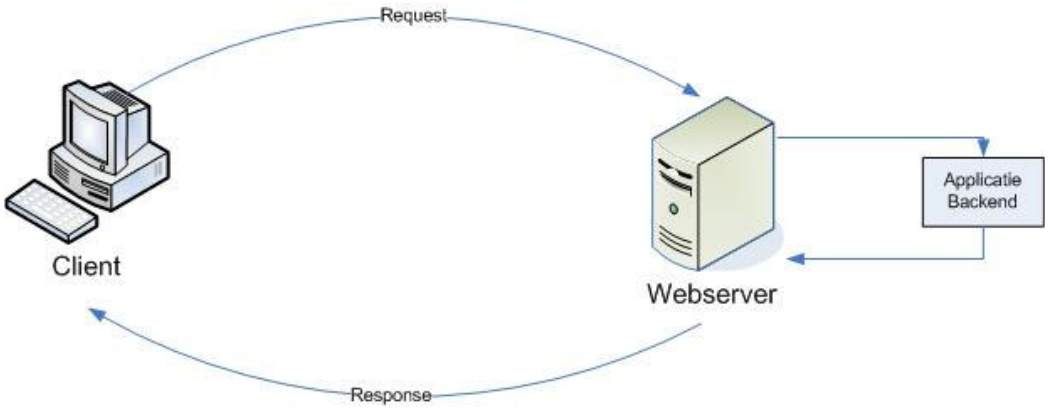
Tijdens deze stap wordt de ontwikkelde methode nogmaals geanalyseerd, en wordt de opgedane ervaring tijdens de casestudy hierin meegenomen. Welke punten van discussie zijn er mogelijk, en welke problemen zijn er naar boven gekomen tijdens de casestudy.

5 Context – Inleiding webframeworks

JavaServer Faces (JSF) is een UI framework gericht op webdevelopment voor het Java platform. JSF is gespecificeerd in JSR-127. Deze specificatie is opgesteld door Sun, in samenwerking met enkele grote bedrijven waaronder Oracle en IBM. De eerste specificatie van JSF is uitgebracht in 2004 maar is vooral de laatste maanden een erg veel besproken onderwerp in magazines en op websites. Dit komt mede doordat JSF onderdeel is van de Java Enterprise Edition 5 (JEE5). Aangezien JSF een standaard is, in tegenstelling tot andere web frameworks, zullen veel bedrijven deze techniek in de komende tijd gaan gebruiken. De specificatie van JSF is zo opgezet, dat deze door verschillende bedrijven geïmplementeerd kan worden. Ook is er een referentie implementatie beschikbaar waarin de basis elementen van JSF geïmplementeerd zijn. De implementaties van bedrijven kunnen de referentie implementatie uitbreiden.
Voor dit rapport is er onderzoek gedaan naar de mogelijkheden en onmogelijkheden van JSF en de verschillende implementaties die er van JSF bestaan. In het rapport wordt advies gegeven over het gebruik van JSF, en er wordt in het bijzonder advies gegeven over het kiezen van een implementatie.

5.1 Ontwikkelen van web applicaties

Het ontwikkelen van een webapplicatie is fundamenteel anders dan het ontwikkelen van een desktop applicatie. Dit komt doordat je hierbij te maken hebt met het HTTP protocol, en met browsers die alleen met HTML om kunnen gaan. De communicatie tussen webserver en client wordt gedaan door “HTTP requests” en “HTTP responses” heen en weer te sturen. Het opvragen van een webpagina gaat zoals weergegeven in Figuur 5-1.



Figuur 5-1 Het request/response model van HTTP

Een groot probleem hierbij is dat HTTP “stateless” is. Dit betekent dat zodra de response van de server naar de client is verstuurd, de webserver de client vergeet. Er is geen verbinding die openblijft tussen de client en de server. Het ontwikkelen van een webapplicatie is door dit model veel tijdrovender dan het ontwikkelen van een desktop applicatie, omdat er veel code moet worden geschreven om de staat van een client te herstellen tussen requests.

5.2 Het gebruik van frameworks

Het op laag niveau werken met het HTTP protocol is erg tijdrovend. Zelfs het ontwikkelen van eenvoudige functionaliteit voor een webapplicatie kan hierdoor erg veel tijd kosten. Om de productiviteit van ontwikkelaars te verhogen worden er frameworks ontwikkeld. Frameworks worden niet alleen gebruikt voor het ontwikkelen van webapplicaties, maar voor bijna elk type applicatie. Een veel gebruikte definitie voor frameworks is: “A framework is a reusable, ``Semi-complete`` application that can be specialized to produce custom applications” [Fayad, Schmidt 1997]. Deze definitie is gebaseerd op een artikel uit 1988 waar frameworks worden gedefinieerd als een “Object-oriented abstract design” [Johnson, Foote 1988]. Het doel van frameworks is dan ook om ontwikkelaars een basis te bieden waarop zij hun applicaties kunnen ontwikkelen. Hierdoor werken ontwikkelaars met een abstractielaag bovenop de eigenlijke techniek, waardoor ze minder lang bezig zijn met het schrijven van hele techniek specifieke code, zoals HTTP communicatie. Dit is erg wenselijk, omdat ontwikkelaars zich op deze manier kunnen concentreren op het ontwikkelen van voor de klant zichtbare functionaliteit. In het geval van webapplicaties is het doel van frameworks om een abstractie bovenop het HTTP protocol te bieden. Een veel gebruikte term bij frameworks is “boiler-plate” code. Dit is code op laag niveau, dat niets te maken heeft met de applicatie waaraan gewerkt wordt, maar noodzakelijk is om met een bepaalde techniek (zoals HTTP) te kunnen werken.

5.3 Webapplicaties met J2EE

Het ontwikkelen van web applicaties is onderdeel van Java Enterprise Edition. Op het moment van schrijven is dit nog J2EE, maar zal vanaf versie 5 JEE gaan heten. Het J2EE / JEE framework biedt een basis voor het ontwikkelen van server-side applicaties zoals WebServices, Enterprise JavaBeans (EJB) en webapplicaties.

Voor het ontwikkelen van webapplicaties worden Servlets en Java Server Pages (JSP) aangeboden. Servlets zijn Java klassen die in een webcontainer leven, en kunnen reageren op HTTP requests. Omdat Servlets bestaan uit Java code, zijn deze minder geschikt voor het genereren van grote stukken HTML. Als dit verwerkt is door de Java code heen is dit erg moeilijk onderhoudbaar.

Om deze reden is JSP ontwikkeld. JSP is een template taal, dat de vorm heeft van HTML. Buiten HTML tags kunnen er echter ook zogenaamde JSP tags opgenomen worden. Deze tags worden vertaald naar Java code, waardoor er dynamische elementen in een JSP pagina kunnen worden opgenomen.

Door Servlets en JSP code te combineren kan er een scheiding worden aangebracht tussen de logica van een applicatie en de weergave hiervan. Het Model-View-Controller design pattern biedt richtlijnen voor het maken van deze scheiding. Een webapplicatie die gebruik maakt van het MVC model wordt een Model2 applicatie genoemd. De meeste webframeworks gebruiken MVC om scheiding tussen de verschillende lagen van de applicatie te realiseren.

5.4 Webframeworks voor Java

Ondanks dat de combinatie van Servlets en JSP een goede basis biedt voor het ontwikkelen van webapplicaties, is er bij het direct werken met deze technieken nog steeds vrij veel code nodig die dicht op het niveau van HTTP zit.

Om invoerwaarden uit formulieren te verwerken bijvoorbeeld, zal een ontwikkelaar de request parameters direct moeten uitlezen. Aangezien de request parameters allemaal als String verstuurd zijn, zullen deze ook eerst moeten worden gecast naar het type wat in het model van de applicatie wordt gebruikt. Door deze afhandeling op laag niveau is het implementeren van een simpel formulier al een behoorlijke klus, terwijl het ontwikkelen van een formulier voor een desktop applicatie vele malen simpeler is.

Om het ontwikkelen van webapplicaties met Servlets en JSP eenvoudiger en sneller te maken zijn er ontelbaar veel frameworks ontwikkeld. Dit is gedaan door commerciële bedrijven, de opensource community, en door vele ontwikkelaars zelf. Veel bedrijven hebben ook hun eigen framework ontwikkeld. De meeste frameworks hebben als primaire functie om een abstractie te bieden bovenop Servlets. Dit kan bijvoorbeeld inhouden dat request parameters door het framework worden opgeslagen in model objecten, zodat hier zonder extra werk op een OO manier mee gewerkt kan worden.

Per framework kan het uiteraard verschillen welke functionaliteit het precies biedt. Grofweg zijn er twee typen webframeworks te onderscheiden: 'Component Based' frameworks en 'Request Based' frameworks [Raible VS 2005]. Het grootste aantal frameworks is te vinden in de tweede categorie. Request Based frameworks gaan nog steeds uit van het werken met de Request en Response van een sessie van de client. Door hier echter een abstractie op te bieden kan het ontwikkelen van applicaties enorm versneld worden. Component Based frameworks gaan een stap verder, en proberen het request/response model volledig te encapsuleren, zodat ontwikkelaars op een volledige component en OO gebaseerde manier kunnen werken. JSF behoort ook tot deze categorie. In onderstaande tabel staat een aantal veel gebruikte frameworks opgesomd.

Request Based frameworks	Component Based frameworks
Struts	Tapestry
WebWork2	Java ServerFaces
Spring WebMVC	Wicket
Maverick	

Struts is verreweg het meest gebruikte framework [Raible VS 2005]. Ondanks dat Struts is ontwikkeld als een open source project, is dit uitgegroeid tot de defacto standaard. Aangezien Struts zoveel gebruikt wordt, zal in dit rapport op sommige punten een vergelijking met Struts gemaakt worden. Velen verwachten met de komst van JSF dat het einde van Struts nadert. Een belangrijke reden voor deze gedachte is dat de hoofdontwikkelaar van Struts heeft meegewerkt aan de JSF specificatie. Het lijkt hierdoor goed mogelijk dat JSF op den duur Struts volledig zal vervangen. Ook zijn er projecten waarbij JSF en Struts gecombineerd worden. Een van de grotere voorbeelden hiervan is het Shale framework. Dit is een subproject van Struts, en heeft als doel de kracht van JSF en Struts te koppelen.

5.4.1 Onmogelijkheden van webapplicaties

Webframeworks kunnen het ontwikkelen van webapplicaties een stuk eenvoudiger maken. Er moet echter wel rekening worden gehouden met het feit dat er uiteindelijk nog steeds wordt gewerkt met het HTTP protocol en webbrowsers. Dit zal dan ook altijd nadelen met zich mee blijven brengen.

Door een abstractie te bieden op request/response model van HTTP, zal sommige functionaliteit van webbrowsers moeilijker bruikbaar worden die wel met traditionele webapplicaties mogelijk zijn. Een voorbeeld hiervan is het gebruik van de "back" en "forward" knoppen en het maken van bookmarks.

De vraag hierbij is of dit te verwijten valt aan het framework, omdat de eigenlijke oorzaak bij het http protocol en bij de webbrowsers ligt. Dit moet dan ook worden onthouden bij het beoordelen van een framework. Zijn bepaalde zaken niet mogelijk doordat het framework slechte ondersteuning biedt, of doordat de techniek (HTTP) wordt gebruikt voor iets wat eigenlijk niet mogelijk is?

Een interessante discussie die hierbij opspeelt is of het wel een goede ontwikkeling is dat er “rich web applicaties” worden ontwikkeld. Zouden deze applicaties niet beter als desktop applicatie kunnen worden ontwikkeld? Deze discussie wordt verder niet behandeld in dit document.

6 Achtergrond

6.1 Definities en context

6.1.1 Context van de definitie van productiviteit

In dit onderzoek wordt er onderzocht hoe de productiviteitswinst die behaald wordt bij het gebruik van een gegeven framework kan worden gemeten. Hierbij hebben we het over productiviteit van een ontwikkelaar die een softwaresysteem moet implementeren. Als we het in dit onderzoek hebben over een framework, wordt hier specifiek gesproken over frameworks gericht op het ontwikkelen van webapplicaties.

6.1.2 Definitie van productiviteit

Om te kunnen meten welke productiviteitswinst een gegeven framework kan bieden, moet er eerst worden vastgesteld wat we precies verstaan onder productiviteit. Bij dit onderzoek is de volgende definitie gebruikt:

“Productiviteit is de snelheid waarmee gespecificeerde functionaliteit van een softwaresysteem geïmplementeerd wordt”.

Deze definitie is gericht op de constructiefase van een software ontwikkeltraject.

6.1.3 Het doel van frameworks

Frameworks bieden een basis structuur, waar nieuwe software bovenop ontwikkeld wordt [Fayad, Schmidt 1997]. Door de basis structuur die een framework biedt, hoeft er minder van deze code zelf ontwikkelt te worden, en kan er effectief meer tijd besteed worden aan het ontwikkelen van voor de klant zichtbare functionaliteit. Het doel van frameworks kan daarmee worden gedefinieerd als:

“Het verhogen van de productiviteit van ontwikkelaars”.

Als er vervolgens wordt gekeken naar de gegeven definitie van productiviteit, kan het doel van frameworks verder worden gespecificeerd als:

“Het doel van een framework is het verhogen van de snelheid waarmee functionaliteit in een software systeem geïmplementeerd kan worden”.

Buiten dit doel, helpt het gebruik van een framework ook code te schrijven volgens een vaste structuur, wat onderhoud eenvoudiger maakt.

6.1.4 Scope van het onderzoek

Binnen dit onderzoek wordt er alleen gekeken naar de hulp die een framework biedt tijdens de constructiefase. De constructiefase is de fase in een project waar het programmeerwerk voor de bouw van het systeem wordt gedaan [McConnell 2004].

Over de onderhoudbaarheid van de code zal wel een kanttekening worden gemaakt (kwantitatieve analyse, zie hoofdstuk 7.5), maar hier zullen geen metingen voor worden uitgevoerd. De onderhoudbaarheid van de code van een systeem is erg belangrijk, aangezien bij veel projecten de onderhoudsfase vele malen langer duurt dan de initiële constructiefase.

Over wat precies onderhoudbare code is zijn de meningen verdeeld. Algemeen kan echter worden genomen dat het volgen van Object georiënteerde best-practices tot goed onderhoudbare code leidt.

6.2 Architectuur assessment

6.2.1 Inleiding

Er zou een voorspelling gedaan kunnen worden over de productiviteitswinst van een framework als er wordt gekeken naar de specificatie en architectuur van het framework. Hier zou duidelijk uit moeten worden welke ondersteuning een framework biedt aan ontwikkelaars, en op welke manier deze ondersteuning geïmplementeerd is. Op basis hiervan kan een verwachting worden geschept over hoe productief er met het framework gewerkt kan worden. Dit geeft echter geen garantie dat dit in de praktijk ook zo is. In de volgende paragraven wordt beschreven hoe daar mee om kan worden gegaan.

6.2.2 Architectuur en de praktijk

Op basis van een assessment van de architectuur van een framework, kan er een bepaalde verwachting worden geschept over wat de productiviteitswinst van het framework zal zijn. Een goede architectuur is echter geen garantie dat er met het framework in de praktijk productief gewerkt kan worden. Als we kijken naar de populariteit van verschillende webframeworks op dit moment, blijkt dat Struts verreweg het meest populair is [Raible VS 2005]. De architectuur van Struts is vergeleken met andere frameworks echter helemaal niet zo netjes, aangezien het bijvoorbeeld een aantal OO best-practices breekt. Een voorbeeld hiervan is dat concrete overerving van framework klassen noodzakelijk is. Ondanks dat heeft Struts populariteit gewonnen doordat hier erg productief mee ontwikkeld kan worden. Dit voorbeeld geeft aan dat een goede architectuur niet per definitie een productiever framework oplevert.

6.2.3 Complexiteit en productiviteit

Als een framework erg veel werk uit handen neemt van een ontwikkelaar, heeft dit in veel gevallen tot gevolg dat het framework complexer wordt. Voor een ervaren ontwikkelaar hoeft dit geen probleem te zijn, maar een onervaren ontwikkelaar kan hier grote problemen mee hebben. In dit geval zal een eenvoudiger framework, waarbij meer werk zelf gedaan moet worden, waarschijnlijk toch productiever zijn voor de onervaren ontwikkelaar. Uitspraken over de productiviteit van een framework moeten daarom wel gekoppeld worden aan de ervaring van de ontwikkelaar waarvoor de uitspraak geldt.

6.3 Meten van productiviteit – Related work

6.3.1 Meten van de productiviteit van ontwikkelaars

Er is veel onderzoek gedaan naar het meten van productiviteit van ontwikkelaars. Er bestaan dan ook diverse technieken en metrieken om productiviteit te meten. De meest opvallende technieken zijn tijdens het onderzoek onderzocht, en ondergebracht in de hieronder weergegeven tabel. De meettechniek die is gekozen voor de ontwikkelde methode (hoofdstuk 7.8) is gebaseerd op de hier besproken technieken.

Onderdeel	Omschrijving
Metriek	Aantal regels code (Lines Of Code)
Gedachte achter de metriek	Meer regels code betekent dat er veel werk gedaan is, daarmee kan worden gezegd dat de ontwikkelaar productief is geweest.

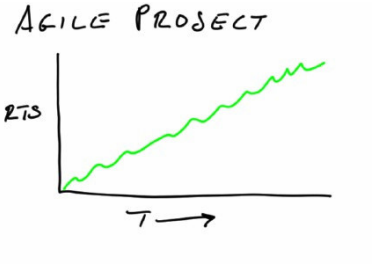
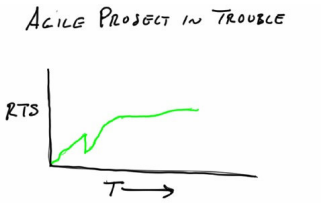
Problemen met de metriek	Slechte code levert vaak meer code op, bijvoorbeeld door code duplicatie. Dit zou bij het gebruik van deze metriek wijzen op een hogere productiviteit, terwijl dit bij onderhoud aan de code grote problemen kan opleveren. [Fowler 2003][McConnell 2004] Buiten dit probleem zijn er een aantal andere zaken waardoor de metriek niet bruikbaar is voor het meten van productiviteit: • Er wordt niet gecontroleerd of de regels code geen fouten bevatten. • Er wordt geen rekening gehouden met de complexiteit van de code. • Er wordt niet gekeken of de regels code iets nuttigs doen, misschien is er functionaliteit geïmplementeerd die voor de klant onbruikbaar is [Fowler 2003]. Dit is echter voor een deel een requirements engineering probleem, en geen constructie probleem.
Andere toepassingen	Het meten van LOC kan bruikbaar zijn om globaal aan te geven hoe groot een systeem is. Als hierbij rekening wordt gehouden met de gebruikte taal kan er een idee worden gevormd hoe groot een systeem is, wat bijvoorbeeld belangrijk is bij het starten van een onderhoudstraject. Dit zegt echter helemaal niets over hoe goed of slecht het systeem is opgezet, en kan daarom ook niet worden gebruikt voor het schatten van kosten voor een onderhoudstraject. Bij het vergelijken van code kan het tellen van LOCs iets zeggen over welke code netter is geschreven. Minder regels code betekent in principe eenvoudiger onderhoud, mits de code niet onnodig complex wordt. Kortere code wijst op een netter design [McConnell 2004]. Hierbij moet de te vergelijken code wel geschreven zijn in dezelfde programmeertaal en met een vergelijkbare codeer stijl.
Bruikbaarheid voor dit onderzoek	De metriek is niet bruikbaar in de zin dat meer regels code een hogere productiviteit voorstellen. De metriek is wel bruikbaar om te vergelijken hoeveel handgeschreven code er nodig is om met gebruik van een gegeven framework gespecificeerde functionaliteit te implementeren, vergeleken met een ander framework. Minder regels code kan worden veroorzaakt doordat het framework meer hulp biedt. Daarbij is er ook minder code dat onderhouden hoeft te worden, en kan daardoor worden gezien als een voordeel.

Onderdeel	Omschrijving
Metriek	LOC / aantal bugs
Gedachte achter de metriek	Minder bugs per regel code betekent dat er productief is gewerkt. Bij deze metriek wordt er nog steeds vanuit gegaan dat meer regels code betekent dat er veel werk is gedaan, maar hierbij wordt rekening gehouden met hoeveel fouten er in deze code zitten.
Problemen met de metriek	• Slechte code levert vaak meer code op [Fowler 2003], bijvoorbeeld door code duplicatie. Door meer regels code neemt het aantal bugs misschien per regel af, terwijl dit niets zegt over hoe zinvol de code is, en wat de kwaliteit hiervan is.

	• Er wordt geen rekening gehouden met de complexiteit van de code. Een zeer complex algoritme zal minder regels code maar eerder fouten bevatten dan een eenvoudig scherm, terwijl dit niets zegt over productiviteit. • Er wordt niet gekeken of de regels code iets nuttigs doen, misschien is er functionaliteit geïmplementeerd die voor de klant onbruikbaar is. Dit hoeft echter geen programmeerfout te zijn.
Andere toepassingen	Als er code gemeten wordt dat voldoet aan een gestelde code standaard, kan de metriek iets zeggen over hoe foutloos een ontwikkelaar werkt. Hierbij moet echter ook rekening worden gehouden met hoe complex de geïmplementeerde functionaliteit is.
Bruikbaarheid voor dit onderzoek	In dit onderzoek worden er alleen metingen gedaan over code die werkt zoals gespecificeerd. Aangezien er geen acceptatie tests door gebruikers kunnen worden gedaan, zullen er ook geen bugs gerapporteerd worden nadat het werk afgerond is. Deze metriek wordt dan ook niet gebruik.

Onderdeel	Omschrijving
Metriek	Functie punten analyse
Gedachte achter de metriek	Het waarderen van functionaliteit van een systeem aan de hand van de complexiteit hiervan, uitgedrukt in getallen. Ook zaken zoals performance eisen worden hierover berekend. Op deze manier kan er taalonafhankelijk worden berekend hoeveel tijd het kost om een systeem te implementeren. De productiviteit van een individuele ontwikkelaar kan vervolgens worden bepaald door te meten hoeveel functie punten er per dag worden geïmplementeerd. Dit kan, omdat de complexiteit van functionaliteit al is berekend bij het tellen van de functiepunten.
Problemen met de metriek	• Het tellen van functiepunten is niet direct vanzelfsprekend. Er is training nodig om dit te kunnen doen. Dit maakt de techniek slecht te gebruiken in situaties waarin deelnemers in het project hier niet bekend mee zijn. • Zaken als hergebruik van code wordt niet berekend in de FP berekening. Vijf soortgelijke functionaliteiten kunnen misschien 20 FP zijn, terwijl 3 totaal verschillende functionaliteiten 10 FP. Bij een goede opbouw van de code zal het minder tijd kosten de vijf soortgelijke functionaliteiten te implementeren, terwijl dit niet is af te lezen aan het aantal functie punten [Fowler 2003]. • Door de complexiteit van het tellen van functiepunten, gebeurt het vaak dat soortgelijke systemen totaal anders beoordeeld worden. De vraag is daarmee in hoeverre een berekend aantal functiepunten betrouwbaar is [Fowler 2003]. • Ondanks dat functiepunten techniek onafhankelijk zijn, zal het implementeren van bepaalde functionaliteit bij het gebruik van verschillende techniek meer of minder inspanning kosten.
Andere toepassingen	Sommige mensen, zoals Martin Fowler, geloven dat het tellen van functiepunten nog niet voldoende is. Volgens hen moet er berekend worden hoeveel waarde een functionaliteit voor de klant heeft. Dit is wellicht een goed idee om te meten hoe productief een team voor een klant is, maar minder op het moment dat er al een functioneel ontwerp bestaat wat simpelweg geïmplementeerd moet worden.
Bruikbaarheid voor dit	Doordat functiepunten te weinig zeggend zijn voor mensen die hier

onderzoek	niet bekend mee zijn, zullen er geen functiepunten gebruikt worden bij dit onderzoek. Een functiepunt analyse zou wel bruikbaar zijn, om aan te geven hoeveel functiepunten per tijdseenheid er ontwikkeld kunnen worden bij het gebruik van een bepaald framework.

Onderdeel	Omschrijving
Metriek	Running Tested Features (RTF)
Gedachte achter de metriek	Deze metriek komt vanuit agile project methoden. 1 Running: De functionaliteit is geïntegreerd in een werkende build. 2 Tested: Er zijn automatisch uit te voeren tests, die bewijzen dat de functionaliteit werkt volgens specificatie. 3 Feature: Voor een eindgebruiker zichtbare functionaliteit, zoals benoemt in een user-story. User-stories komen uit de Extreme Programming methodiek [Beck 2005]. De gedachte achter deze metriek is dat het doel van een software development traject is om zoveel mogelijk werkende functionaliteit voor een zo laag mogelijke prijs op te leveren. Door het meten van geïntegreerde en geteste functionaliteit, kan er worden bepaald hoeveel functionaliteit er per week door een team of individu kan worden opgeleverd. Bij een agile project, zou dit een constant stijgende lijn moeten zijn zoals weergegeven in figuur Figuur 6-1.  *Figuur 6-1* Hierbij wordt voorkomen dat er code van slechte kwaliteit wordt opgeleverd. Als dit namelijk wel gebeurt, zal dit later in het project een productiviteitsdaling opleveren zoals weergegeven in figuur Figuur 6-2.  *Figuur 6-2* Bij een agile project wordt er op elk moment refactoring toegepast, waardoor het implementeren van functionaliteit niet stopgezet hoeft te worden om grootschalig onderhoud aan de code te doen. Dit gebeurt wel in de meeste niet-agile projecten.

Problemen met de metriek	Er is geen duidelijke definitie hoeveel functionaliteit er in een enkele user-story kan worden ondergebracht. Het wordt aanbevolen om user-stories zo klein te houden dat ze binnen een enkele iteratie te implementeren zijn.
Andere toepassingen	
Bruikbaarheid voor dit onderzoek	Tijdens het onderzoek is deze metriek gebruikt voor het meten van productiviteit. De reden hiervoor is dat de te implementeren functionaliteit van te voren is opgedeeld in kleine stukken in de vorm van use-cases. Vervolgens wordt er gemeten hoe lang er over gedaan wordt om de beschreven functionaliteit in een use-case te implementeren. Dat betekent dat de functionaliteit Running en Tested moet zijn. Als dit per use-case wordt gedaan, kan er vervolgens aan het eind van het traject worden berekend hoeveel tijd het kost om de beschreven functionaliteit te implementeren. Dit kan direct worden vergeleken met een meting die gedaan is bij het gebruik van een ander framework.

6.4 Persoonlijke verschillen

6.4.1 Inleiding

Diverse onderzoeken hebben uitgewezen dat de productiviteit van individuele ontwikkelaars onderling enorm kan verschillen. Een belangrijke factor hierbij is uiteraard opleiding en ervaring. Uit verschillende onderzoeken blijkt echter dat ook ontwikkelaars met gelijke opleiding en ervaring enorm verschillend kunnen presteren.

Een klassiek onderzoek naar dit onderwerp is het onderzoek van Sackman, Erikson en Grant uit 1968 [Sackman 1968]. In dit onderzoek werd het verschil in productiviteit tussen off-line en on-line ontwikkelen onderzocht. Uit het onderzoek bleek echter ook dat er enorme onderlinge prestatie verschillen tussen de individuele ontwikkelaars bestonden, onafhankelijk van de gebruikte techniek. Dit was voor het schrijven van code een factor 20:1 en bij debuggen zelfs 28:1. Dit is ook de reden dat dit artikel nog steeds vaak als referentie wordt gebruikt.

De gegeven getallen (20:1, 28:1) zijn volgens diverse onderzoekers echter onjuist. Lutz Prechelt [Prechelt 1999] laat in zijn paper zien dat er statistische fouten in het onderzoek zitten, waardoor de cijfers veel hoger lijken dan dat ze in werkelijkheid zijn. Prechelt heeft vervolgens een veel grootschaliger onderzoek gedaan waarbij de productiviteitsverschillen tussen ontwikkelaars wordt gemeten. Prechelt, en vele anderen (Curtis 1981, Mills 1983, DeMarco and Lister 1985, Curtis et al. 1986, Card 1987, Boehm and Papaccio 1988, Valett and McGarry 1989, Boehm et al. 2000) komen bij hun onderzoeken echter nog steeds tot de conclusie dat er enorme productiviteitsverschillen tussen ontwikkelaars kunnen zijn, zelfs als opleiding en aantal jaar ervaring vergelijkbaar is.

6.4.2 Werkomgeving

Ook de omstandigheden waarin ontwikkelaars werken hebben invloed op hun productiviteit. Uit een onderzoek van DeMarco en Lister [DeMarco, Lister 1985] blijkt dat de best presterende ontwikkelaars meestal over een grotere, rustigere werkruimte beschikken dan de ontwikkelaars die duidelijk slechter werken. Hieruit concluderen zij dat er een duidelijk verband is tussen productiviteit en de kwaliteit van de werkruimte.

6.4.3 Belang voor dit onderzoek

De hier beschreven onderzoeken zijn zeer belangrijk bij het ontwikkelen van een methode voor het meten van productiviteit. Door de grote onderlinge productiviteitsverschillen tussen

ontwikkelaars is het onmogelijk om conclusies te trekken uit de snelheid waarmee een enkele ontwikkelaar een applicatie kan implementeren.

7 Framework Productivity Measurement Method

7.1 Inleiding

7.1.1 Doel van de methode

Het doel van de tijdens het onderzoek ontwikkelde methode, is het meetbaar maken van de productiviteitswinst van een framework. Zodoende kan een framework op basis van een vergelijking met een referentie framework geëvalueerd worden. Dit kan gebruikt worden bij het kiezen van een webframework in een software ontwikkeltraject.

7.1.2 Definitie van productiviteitswinst

In paragraaf 6.1.2 is de volgende definitie gegeven voor productiviteit:

“Productiviteit is de snelheid waarmee gespecificeerde functionaliteit van een softwaresysteem geïmplementeerd wordt”.

Als we het hebben over de productiviteitswinst dat een framework biedt, betekent dit het verschil in productiviteit tussen het gebruik van het gegeven framework, en de productiviteit wanneer het framework niet wordt gebruikt.

Omdat het ontwikkelen van webapplicaties zonder het gebruik van een framework in de meeste gevallen geen goede keuze is (paragraaf 5.4), wordt er bij de ontwikkelde methode echter gebruik gemaakt van een referentie framework waarmee vergeleken wordt.

7.1.3 Introductie methode

Zoals besproken in hoofdstuk 6.3 is het niet eenvoudig om de productiviteit van ontwikkelaars te meten. Hierdoor is het erg lastig om iets te kunnen zeggen over de productiviteitswinst dat een webframework biedt bij het ontwikkelen van webapplicaties. In veel gevallen zullen uitspraken hierover gebaseerd zijn op een gevoel, en niet op gemeten resultaten.

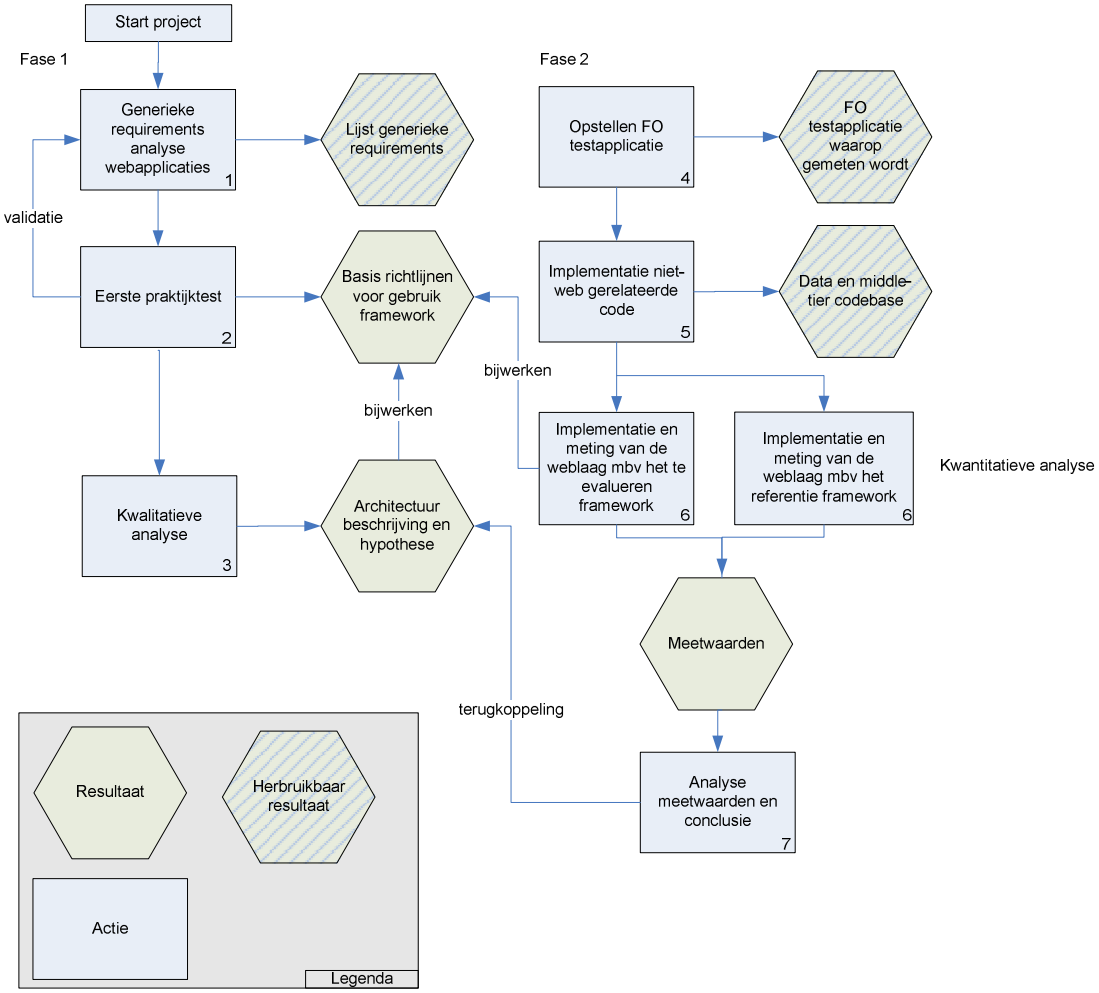
De ontwikkelde methode, de “Framework Productivitity Measurement Method” (FPMM) is er op gericht om de productiviteitswinst meetbaar te maken. FPMM is zodanig ontwikkeld dat factoren die invloed op de productiviteit hebben, maar onafhankelijk van het te onderzoeken framework zijn, een zo minimaal mogelijke invloed hebben op de resultaten.

7.2 De FPMM methode

FPMM bestaat uit een aantal stappen, onderverdeeld in twee fases. Dit traject is weergegeven in
Figuur 7-1. Elke stap levert een resultaat op. Dit resultaat wordt gebruikt in latere stappen.

De eerste fase bestaat uit het doen van vooronderzoek en een kwalitatieve analyse. Het doel van deze fase is om basis ervaring met het framework op te doen, en een hypothese te kunnen maken met betrekking tot de verwachte productiviteitswinst. In de tweede fase worden er daadwerkelijke metingen met betrekking tot de productiviteit gedaan. Hier kan uit geconcludeerd worden of het framework wel of niet en goede keuze is, als hierbij de nadruk ligt op de te halen productiviteit. De metingen worden uitgevoerd tijdens het implementeren van een testapplicatie.

In de rest van dit hoofdstuk wordt elke stap afzonderlijk besproken.



Figuur 7-1 De verschillende stappen van FPMM

7.3 Generieke requirements analyse webapplicaties (stap 1)

7.3.1 Doel generieke requirements

Het doel van het opstellen van generieke requirements is het verkrijgen van inzicht in welke ondersteuning van het framework belangrijk is. Op basis van deze requirements zullen ook de metingen worden uitgevoerd.

7.3.2 Vaststellen scope

Om iets te kunnen zeggen over de productiviteit van een framework, moet er eerst worden vastgesteld binnen welke scope dit wordt bekeken. Welke ondersteuning moet een framework bieden om ontwikkelaars productiever te maken? Ten eerste moet er worden vastgesteld voor welk type applicatie het framework wordt geëvalueerd. Het is namelijk goed mogelijk dat een

framework zeer goed bruikbaar is voor het ontwikkelen van een bepaald type applicatie, maar minder bruikbaar voor een ander type applicatie. Voorbeelden van type applicaties zijn intranet applicaties voor de financiële markt, communities en bedrijfsportalen. Verschillen tussen de verschillende type applicaties zijn bijvoorbeeld het belang van het uiterlijk van de website en de hoeveelheid invoerformulieren.

7.3.3 Opstellen generieke requirements

Vervolgens moet er op basis van dat type applicatie een lijst worden opgesteld met veel voorkomende functionaliteit binnen dat type applicatie. Dit zijn generieke requirements, waar het framework ondersteuning voor moet bieden. Deze lijst moet worden opgesteld in samenwerking met een expert op het gebied van dat type applicatie.

7.3.4 Herbruikbaarheid generieke requirements

De lijst met generieke requirements is tot op zekere hoogte herbruikbaar voor het evalueren van meerdere webframeworks. De voorwaarde die hierbij gesteld wordt is dat de scope van het type applicatie waarvoor de evaluatie wordt uitgevoerd hetzelfde is. Als dit niet het geval is, kan de lijst generieke requirements niet zondermeer hergebruikt worden. In de meeste gevallen zullen de hier voorgestelde generieke requirements in ieder geval een goede basis vormen.

7.4 Eerste praktijktest (stap 2)

7.4.1 Doel eerste test

Voorafgaand aan het implementeren van de testapplicatie waarbij metingen worden uitgevoerd, moet er een kleine test worden geïmplementeerd. Het doel hiervan is het opdoen van basiservaring met het framework.

7.4.2 Implementatie

Zoals gezegd is het doel van de eerste test om ervaring met het framework op te doen. Wat er precies aan functionaliteit wordt geïmplementeerd tijdens deze test is daarom niet belangrijk. Het is wel aan te raden om functionaliteit te implementeren dat aansluit op de opgestelde generieke requirements.

7.5 Kwalitatieve analyse (stap 3)

7.5.1 Doel van de kwalitatieve analyse

Het uitvoeren van de kwalitatieve analyse heeft twee doelen. Het eerste doel is om basis kennis en ervaring op te doen van het framework, zodat het uitvoeren van de kwantitatieve analyse minder problemen oplevert. Het tweede doel is om een hypothese te kunnen maken over de verwachte productiviteitswinst, de leercurve van het framework en de onderhoudbaarheid van framework specifieke code.

7.5.2 Uitvoer kwalitatieve analyse

De kwalitatieve analyse bestaat uit een aantal onderdelen:

- Onderzoeken functionaliteit
- Assessment architectuur
- Onderzoeken tooling en ondersteuning

1. Onderzoeken functionaliteit
Hierbij wordt er gekeken naar welke functionaliteit het framework biedt volgens de specificatie of documentatie hiervan. Dit heeft als doel om een beeld te vormen van de mogelijkheden van het framework. Ook zal hierbij duidelijk worden welke problemen het framework op zou moeten lossen.

2. Assessment architectuur
Het tweede onderdeel van de kwalitatieve analyse is het doen van een assessment op de architectuur van het framework. Dit heeft als doel om uitspraken te kunnen doen over welke invloed de architectuur op de zelf te ontwikkelen code heeft, hoeveel configuratie er noodzakelijk is en hoe flexibel het framework is. Hierbij kan er in elk geval worden gekeken naar de zaken beschreven in Tabel 7-1.

Criteria	Omschrijving
Hergebruik interface componenten	Interface componenten moeten zoveel mogelijk kunnen worden hergebruikt. Ideaal is het als een component zoals een poll kan worden ontwikkeld, en vervolgens in meerdere applicaties gebruikt kan worden.
Vrijheid in vormgeving gebruikersinterface	Het gebruik van een framework zou geen grote beperkingen voor het uiterlijk van een applicatie mogen hebben. Het uiterlijk van webapplicaties kan, afhankelijk van het type applicatie, erg belangrijk zijn. Grafische ontwerpers moeten zoveel mogelijk vrijheid hebben, zonder door het framework beperkt te worden.
Hergebruik code	Het hergebruiken van code wordt door velen gezien als een best-practice. Afhankelijk van de opzet van een framework zal dit meer of minder moeite kosten.
Duidelijke scheiding code en opmaak	Het uiterlijk van webapplicaties is vaak erg veranderlijk. Het is daarom wenselijk dat opmaak zoveel mogelijk losgekoppeld is van de onderliggende code. Dit is erg belangrijk voor de onderhoudbaarheid van een webapplicatie.
Ondersteunen Object Georiënteerd programmeren	Een probleem bij het werken met request parameters, is dat dit altijd String waarden zijn. Hierbij moet er zelf extra werk verzet worden om object georiënteerd te werken. Ook het gebruik van bepaalde design patterns binnen het framework kunnen de code gestructureerder maken, en hiermee beter onderhoudbaar.

Tabel 7-1

3. Onderzoeken tooling en ondersteuning
Tooling kan grote invloed uitoefenen op de productiviteit van een ontwikkelaar. De vraag is hierbij in hoeverre tooling wel of niet framework onafhankelijk is, en in hoeverre een framework tooling onafhankelijk is. Afhankelijk van de architectuur van een framework zal ondersteuning door tooling meer of minder mogelijk zijn. Hierdoor is de beschikbare tooling voor een deel afhankelijk van de mogelijkheden van het framework. Om deze reden wordt de beschikbare tooling meegenomen in het onderzoek naar een framework.

Ondersteuning voor een framework in de vorm van documentatie, een actieve gebruikersgroep, en actieve ontwikkeling aan een framework zijn erg belangrijk voor de leercurve en het voortbestaan van een framework. In Tabel 7-2 is een aantal criteria beschreven waar een framework op beoordeeld kan worden in deze categorie.

Criteria	Omschrijving
IDE ondersteuning	Afhankelijk van hoe het framework in elkaar zit zal er meer of minder IDE ondersteuning mogelijk zijn. Buiten de mogelijkheden is het ook belangrijk om te kijken hoe de daadwerkelijke ondersteuning is.
Documentatie	Goede documentatie is onontbeerlijk bij het gebruik van een

	framework. Als er geen goede documentatie beschikbaar is zal de leercurve veel steiler zijn, dan wanneer er wel goede documentatie is.
Ondersteunende Gebruikersgroep	Bij het oplossen van problemen is het altijd handig als er een grote gebruikersgroep is. De kans is dan groot dat er hulp is te vinden op bijvoorbeeld forums. Ook zal een framework met een enorme gebruikersgroep meestal beter onderhouden worden.
Actieve ontwikkeling	Het is goed als er actief aan het framework doorontwikkeld wordt. Dit hoeft niet te betekenen dat er alleen maar nieuwe functionaliteit wordt toegevoegd, maar ook dat bugs in het framework snel worden aangepakt.

Tabel 7-2

In deze fase wordt er onderzocht wat de beste set van tooling is dat op dat moment beschikbaar is. Met behulp van deze tooling worden de testapplicaties ontwikkeld. Een alternatief hierop is het kiezen van een standaard set van tooling, onafhankelijk van of deze tooling ondersteuning voor het framework biedt. Deze methode is een goede keuze als een bedrijf bijvoorbeeld afhankelijk is van een beperkte set tooling.

7.5.3 Richtlijnen gebruik framework

Na het uitvoeren van de eerste praktijktest en de kwalitatieve analyse moet er een goed beeld gevormd zijn over hoe het framework in elkaar zit, en hoe het framework het beste gebruikt kan worden. Als bijproduct van dit onderzoek kunnen er daarom richtlijnen worden opgesteld voor het gebruik van het framework. In de richtlijnen kan worden opgenomen hoe bepaalde standaard problemen met behulp van het framework kunnen worden opgelost. Deze richtlijnen worden verder niet gebruikt tijdens de onderzoeksmethode, maar kunnen erg waardevol zijn als het framework in gebruik wordt genomen door een ontwikkelteam.

7.6 Opstellen FO testapplicatie (stap 4)

7.6.1 Opstellen functioneel ontwerp

De productiviteitswinst van een framework wordt gemeten tijdens het implementeren van een testapplicatie. Deze testapplicatie moet worden beschreven in een functioneel ontwerp. Het functioneel ontwerp kan worden geschreven in de vorm van use-cases. Op deze manier is er duidelijk onderscheid te maken tussen de verschillende onderdelen van de applicatie. Binnen een use-case kan onderscheid worden gemaakt tussen de verschillende functionaliteiten die samen de use-case vormen. Andere technieken voor het vastleggen van het functioneel ontwerp zijn eventueel mogelijk. Dit wordt echter niet binnen dit onderzoek gebruikt.
Het detail niveau van het functioneel ontwerp moet dusdanig zijn dat ontwikkelaars niet meer zelf hoeven na te denken op welke manier een beschreven functionaliteit exact moet werken (functioneel gezien). Dit zou namelijk invloed hebben op de productiviteit, zonder dat dit afhankelijk is van het te evalueren framework.

7.6.2 Richtlijnen voor de testapplicatie

De testapplicatie moet zoveel mogelijk alle generieke requirements implementeren die gelden voor het type applicatie waarop de scope is vastgesteld. Dit om te voorkomen dat er alleen functionaliteit wordt getest waar het framework erg sterk of zwak in is.
De testapplicatie moet voldoende groot zijn om de testresultaten niet afhankelijk te laten zijn van te kleine verschillen. De implementatie van de applicatie moet worden verspreid over meerdere dagen, om te voorkomen dat de resultaten beïnvloed worden doordat de ontwikkelaar een slechte dag heeft. Dit risico wordt verminderd op het moment dat er op verschillende dagen en tijdstippen gewerkt wordt.

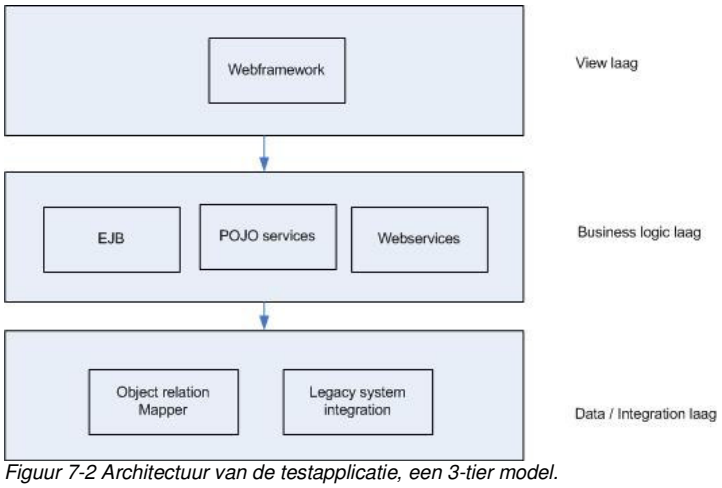
7.6.3 Overlappende Use-cases

Tijdens het opstellen van het functioneel ontwerp voor de testapplicatie moet er een afweging worden gemaakt tussen validatie en beschikbare tijd. Door overlappende use-cases op te stellen wordt soortgelijke functionaliteit meerdere keren getest. Naar mate er meer bewijs is voor een resultaat, wordt het resultaat betrouwbaarder. Overlappende use-cases dienen hierbij als validatie voor de resultaten. Het implementeren van extra use-cases kost uiteraard (veel) tijd. Per evalutatie traject zal er daarom moeten worden bepaald hoeveel validatie noodzakelijk is.

7.7 Implementatie niet-web gerelateerde code (stap 5)

7.7.1 Inleiding

De meeste webframeworks richten zich, als we het hebben over een typische 3-tier applicatie [Johnson 2002], geheel op de view laag. Dit model is weergegeven in Figuur 7-2.



Figuur 7-2 Architectuur van de testapplicatie, een 3-tier model.

Door dit model is het goed mogelijk om de twee implementaties van de testapplicatie te baseren op dezelfde codebase voor de data/integration laag en business logic laag. Dit scheelt uiteraard ontwikkeltijd, en zoals wordt besproken in paragraaf 7.8.6, helpt dit bij het isoleren van de framework specifieke code.

7.7.2 Uitwerking

De Business logic laag en de data/integration laag kan voorafgaand aan de implementatie van de view/web laag worden ontwikkeld. Als deze code goed getest wordt, hoeft er tijdens het implementeren van de weblaag niets meer te worden veranderd aan de onderliggende lagen. Op deze manier kan de aandacht tijdens het uitvoeren van de metingen volledig worden gericht op de weblaag.

7.8 Uitvoeren metingen – Kwantitatieve analyse (stap 6)

7.8.1 Inleiding

Bij de kwantitatieve analyse wordt er een testapplicatie geïmplementeerd. Tijdens het ontwikkelen hiervan worden er tijdsmetingen gedaan. De meetresultaten worden vervolgens gebruikt om te bepalen wat de productiviteitswinst is bij het gebruik van het geteste framework.

Aangezien deze resultaten onafhankelijk moeten zijn van de ontwikkelaar die de testapplicatie heeft geïmplementeerd moet er worden gemeten volgens een aantal richtlijnen.

7.8.2 Omgaan met persoonlijke verschillen

In paragraaf 6.4 is beschreven dat er grote verschillen kunnen zijn tussen de productiviteit van verschillende individuen. Om uitspraken te kunnen doen over de productiviteitswinst van een framework moeten de metingen zodanig worden uitgevoerd dat de persoonsgebonden productiviteitsverschillen hier geen invloed op hebben.

Om deze reden worden de metingen gebaseerd op het verschil tussen de productiviteit bij het gebruik van het te evalueren framework, en de productiviteit bij het gebruik van een referentie framework.

Om dit uit te voeren wordt de testapplicatie tweemaal geïmplementeerd, eenmaal met het te evalueren framework, en eenmaal met het referentie framework. Op deze manier kan er een vergelijking worden gemaakt tussen het te evalueren framework en het referentie framework.

De gedachte achter deze methode is dat het procentuele verschil in productiviteit, bij gebruik van de verschillende frameworks, hetzelfde blijft naarmate een ontwikkelaar meer of minder ervaring heeft. Hier is één uitzondering op. Deze uitzondering wordt besproken in paragraaf 6.2.3.

7.8.3 Gekozen meetmethode

Om ontwikkeltijd te koppelen aan geïmplementeerde functionaliteit wordt er gebruik gemaakt van de Running Tested Features methode.

Deze methode is besproken in paragraaf 6.3.1. De reden voor deze keuze is dat het de meest eenvoudige metriek is die niet gebaseerd is op het tellen van regels code. Zoals besproken in paragraaf 6.3.1 levert het tellen van regels code geen beeld op van de productiviteit van een ontwikkelaar.

7.8.4 Alternatieven

Het doen van een functiepunt analyse maakt het onderzoek complexer, terwijl dit op dit moment geen toegevoegde waarde heeft. In een toekomstig onderzoek zou dit echter wel in de methode kunnen worden opgenomen. Het is namelijk ideaal als er bij kostenschattingen berekeningen gedaan kunnen worden zoals: “Bij het gebruik van dit framework, kost het implementeren van N functiepunten, T tijd.”. Hier staan echter twee problemen in de weg.

Ten eerste is er geen garantie dat functiepunt analyses altijd dezelfde resultaten opleveren, ondanks dat er duidelijke richtlijnen zijn hoe een functiepunt analyse moet worden uitgevoerd. Martin Fowler [Fowler 2003] merkt op dat functiepunt analyses uitgevoerd door verschillende personen volledig afwijkende resultaten opleveren. Chris Verhoef [Verhoef 2005] daarentegen schrijft dat getrainde functiepunt tellers zeer consistente resultaten behalen.

Info Support gebruikt functiepunt analyses bij het maken van kostenschattingen voor projecten waar de ontwikkelstraat van Info Support wordt gebruikt. Op basis van cijfers uit uitgevoerde projecten en goede scoping van het type project, kunnen er zeer exacte functiepunt analyses gedaan worden. In een situatie als deze zou de keuze van een framework hierin meegenomen kunnen worden. Er zouden dan schattingen gedaan kunnen worden zoals: “Bij het gebruik van framework X, kunnen er N functiepunten per uur worden geïmplementeerd”.

7.8.5 Het meten van tijd

In FPMM wordt tijdens het implementeren van de testapplicatie, gemeten hoeveel tijd hieraan besteed wordt. Om te voorkomen dat de resultaten worden beïnvloed door gebrek aan ervaring met het framework wordt de tijd die besteed wordt aan het oplossen van fouten en het uitzoeken van oplossingen niet meegeteld. De gemeten tijd kan daarmee als volgt worden beschreven:

Meetwaarde = ontwikkeltijd – (debugtijd + uitzoektijd)

Debuggen is een onderdeel van het constructie proces wat bij elk project een groot deel van de ontwikkeltijd in beslag zal nemen. McConnell schrijft zelfs dat debuggen het grootste deel van de ontwikkeltijd in beslag neemt in de meeste projecten [McConnell]. Hoeveel tijd debuggen kost hangt echter sterk af van de ervaring van een ontwikkelaar met een bepaalde techniek. Naar mate een ontwikkelaar meer ervaring opdoet met een bepaalde techniek (of framework) zal er minder tijd aan het oplossen van fouten worden besteed.

De keuze voor de adoptie van een framework is in veel gevallen een strategische keuze binnen een bedrijf of ontwikkelteam. Het framework zal dan ook voor langere tijd in gebruik worden genomen. Om deze reden is het vooral belangrijk om te weten hoe productief er met het framework kan worden gewerkt, nadat ontwikkelaars ervaring met het framework hebben opgedaan. Door de debugtijd en uitzoektijd niet mee te nemen in de tijdsmeting kan er een beeld gevormd worden over de productiviteitswinst van het framework, onafhankelijk van de ervaring die ontwikkelaars hebben met het framework.

De gebruikte debugtijd wordt wel als aparte meetwaarde vermeld, om zodoende een beeld te geven van de werkelijke productiviteit (inclusief debugtijd) voor een ontwikkelaar met slechts minimale ervaring met het framework. Dit kan ook een indicatie geven van de leercurve van het framework.

Uit praktisch oogpunt is het aan te raden de ontwikkeltijd vast te leggen met behulp van bijvoorbeeld een stopwatch. Doordat een stopwatch eenvoudig is te pauzeren, is het makkelijker om goed onderscheid te maken tussen ontwikkeltijd en debug tijd.

7.8.6 Isoleren framework afhankelijke code

Bij een n-tier applicatie, wat gebruikelijk is voor webapplicaties, is een webframework meestal in de bovenste laag te gebruiken. Meestal wordt deze laag de “view laag” genoemd. Deze laag is verantwoordelijk voor het tonen van de gebruikersinterface aan de client. Tijdens het doen van metingen zou er alleen moeten worden gekeken naar code die afhankelijk is van het gebruikte framework. Als dit niet gedaan wordt zijn er veel meer factoren in de code waar de ontwikkeltijd door beïnvloed wordt. Om deze reden moet er onderscheid worden gemaakt tussen framework afhankelijke code, en framework onafhankelijke code. Om dit mogelijk te maken moet de testapplicatie volgens een n-tier model worden opgezet. Alleen code in de view laag, waar het webframework gebruikt wordt, wordt meegenomen in de metingen. De overige code kan ook worden hergebruikt als de testapplicatie met meerdere webframeworks wordt geïmplementeerd.

7.8.7 Tellen aantal regels code

Naast het meten van de bestede ontwikkeltijd kan ook het aantal benodigde regels code worden geteld. Deze metriek wordt echter niet gebruikt om aan te geven hoe productief een ontwikkelaar is zoals beschreven in hoofdstuk 6.3.1.

De metriek wordt wel gebruikt om een inschatting te maken in het verschil in aantal regels code tussen verschillende implementaties van een testapplicatie. Minder regels code betekent in principe minder werk, en dit geeft een indicatie over het te behalen productiviteitsverschil.

Deze indicatie is niet genoeg om direct conclusies uit te trekken, maar kan wel worden gebruikt als mogelijke verklaring van het verschil in meetwaarden.

Bij het tellen van aantal regels code is het uiteraard van belang dat dit onder bepaalde voorwaarden gebeurt. Er moet worden vastgesteld over welke code de meting wordt gedaan. Hierbij is er gekozen om alleen de code in de view laag, waar het webframework wordt gebruikt, mee te tellen. De meting moet worden gedaan over code met een ongeveer gelijke coding-style. Verschillen in coding-style kan vrij veel uitmaken in het aantal regels. Bij een vergelijking tussen twee implementaties is het daarom belangrijk dat de coding style van beide implementaties ongeveer hetzelfde is. De meting moet worden gedaan met behulp van dezelfde tools. Verschillende tools kunnen verschillende resultaten opleveren bij het tellen van code. Dit is geen probleem, zolang er bij een vergelijking gebruik wordt gemaakt van dezelfde tools.

7.8.8 Omgaan met configuratie

Bij veel frameworks hebben configuratie bestanden een centrale rol. Veel frameworks gebruiken XML bestanden om configuratie in op te slaan. Bij sommige frameworks is dit zover doorgevoerd dat het configureren van de applicatie een groot deel van de totale ontwikkeltijd in beslag neemt. De tijd besteed aan configureren wordt dan ook meegeteld als ontwikkeltijd.

7.9 Meetresultaten en conclusie (stap 7)

7.9.1 Meetwaarden

De resultaten van de metingen die zijn verkregen tijdens het implementeren van de testapplicatie worden weergegeven in tabellen. Het is belangrijk dat er onderscheid wordt gemaakt tussen de verschillende geïmplementeerde functionaliteiten, zodat er kan worden geanalyseerd in welke type functionaliteit het framework goed of slecht presteert. Voor elke use-case uit het functioneel ontwerp van de testapplicatie wordt er een tabel met meetresultaten gevuld. Tabel 7-3 is een template voor deze tabellen.

Naam use-case

Onderdeel	Onderdeel 1	Onderdeel 2	Onderdeel 3	Totaal
Ontwikkeltijd Framework A	Tijd in min.	“”	“”	“”
Debugtijd Framework A	Tijd in min.	“”	“”	“”
Regels code Framework A		“”	“”	“”
Ontwikkeltijk Framework B	Tijd in min.	“”	“”	“”
Debugtijd Framework B	Tijd in min.	“”	“”	“”
Regels code Framework B		“”	“”	“”
Productiviteitsfactor Framework A ten opzichte van Framework B	Ontwikkeltijd framework B / ontwikkeltijd framework A	“”	“”	“”

Tabel 7-3 Template voor een resultaten tabel

7.9.2 Algemene conclusie

Als alle meetgegevens verzameld zijn is het mogelijk hier een algemene lijn in te vinden. Op deze manier kunnen we, gekeken over de gehele breedte van de functionaliteit, zeggen dat framework A, N% productiviteitswinst oplevert, ten opzichte van het gebruikte referentie framework.

Dit wordt gedaan door het gemiddelde te nemen van de gemeten productiviteitsfactor van elke use-case.

Hiermee hebben we een resultaat waarbij het verschil in productiviteit tussen twee frameworks wordt aangetoond.

7.9.3 Sterkte / zwakte analyse

De volgende stap die moet worden genomen is de meetresultaten verder analyseren om vast te kunnen stellen wat de sterke en zwakke punten van het framework zijn. Deze analyse kan gebruikt worden om te bepalen voor welke type functionaliteit het framework een goede keuze is.

Een mogelijkheid hierbij is dat voor een gegeven project een combinatie van het referentie framework en het geëvalueerde framework de beste keuze is. Dit kan zo zijn als beide frameworks op bepaalde punten significante voordelen bieden. Hierbij moet uiteraard rekening worden gehouden met de extra tijd die nodig is om de twee frameworks samen te laten werken.

7.9.4 Inschatting leercurve

Ondanks dat debugtijd niet wordt meegeteld bij het meten van de benodigde tijd voor het implementeren van de test applicatie, wordt de debugtijd wel vastgelegd. Dit kan samen met de resultaten uit de kwalitatieve analyse worden gebruikt om een inschatting te doen van de leercurve van het framework. Tijdens de kwalitatieve analyse is er onderzocht hoe goed de documentatie en ondersteuning van het framework is, en er is een analyse op de architectuur van het framework gedaan.

De analyse op de architectuur van het framework kan een indicatie geven hoe gecompliceerd het framework in elkaar zit. Als vervolgens blijkt dat er erg veel tijd in debuggen is gestoken tijdens het implementeren van de testapplicatie, is dit een indicatie dat de leercurve vrij stijl is. Buiten een indicatie kunnen hier echter geen harde uitspraken over worden gedaan. Dit komt doordat de leercurve erg afhankelijk is van hoeveel ervaring een ontwikkelaar heeft, en of dit ervaring is met een soortgelijk framework.

7.9.5 Het kiezen van een framework

Na het FPMM evaluatie proces te zijn doorlopen, kan er worden geconcludeerd of het geëvalueerde framework een geschikte keuze is of niet. Als hierbij de belangrijkste factor is dat er met het framework productief kan worden gewerkt, zullen de verkregen meetresultaten doorslaggevend zijn.

Dit hoeft echter niet altijd het geval te zijn. Als er een systeem wordt ontwikkeld dat voor langere tijd onderhouden moet worden, is goed onderhoudbare code veel belangrijker dan een iets sneller constructieproces. Het onderhoudsproces is bij dit soort systemen veel omvangrijker dan de initiële bouw. Als er met een bepaald framework met een factor twee keer zo snel ontwikkeld kan worden, maar slecht onderhoudbare code oplevert, is het een slechte keuze om dit framework te gaan gebruiken.

Bij een project waarbij zeer snel iets gerealiseerd moet worden, kan het een slechte keuze zijn om een framework te kiezen met een steile leercurve, als de ontwikkelaars nog geen ervaring met dit framework hebben.

Dit zijn afwegingen die moeten worden gemaakt bij het kiezen van framework, in het bijzonder als dit een strategische keuze binnen een bedrijf is. De kwalitatieve analyse en de verkregen meetresultaten zullen samen moeten bepalen hoe geschikt het framework is voor de gegeven situatie.

7.10 Samenvatting

In dit hoofdstuk is FPMM voorgesteld, en zijn de verschillende stappen van de methode beschreven. In het volgende hoofdstuk zal FPMM worden toegepast in een praktijk situatie, waarbij het JSF framework wordt geëvalueerd met behulp van FPMM.

8 Case studie – Het JSF framework

8.1 Inleiding

In dit hoofdstuk wordt FPMM toegepast in een praktijk situatie waarbij het JSF framework wordt geëvalueerd. Deze casestudy is te gebruiken als voorbeeld van hoe FPMM in de praktijk kan worden toegepast. De indeling van dit hoofdstuk is volgens de stappen van FPMM.

8.2 Generieke requirements analyze webapplicaties (stap 1)

8.2.1 Inleiding

Zoals besproken in hoofdstuk 7.3 wordt er aan het begin van een evaluatie traject een lijst opgesteld met generieke requirements voor het type applicatie waarvoor het framework geëvalueerd wordt.

8.2.2 Scope

Bij de case studie is de scope van het type applicatie waar het JSF framework voor wordt geëvalueerd vastgesteld op intranet applicaties zoals die bijvoorbeeld in de financiële wereld worden gebruikt. Typerend aan deze applicaties is dat er veel gegevens ingevoerd moeten worden door gebruikers, en de uiterlijke representatie meestal van ondergeschikt belang is.

8.2.3 Lijst met generieke requirements

Criteria	Omschrijving
Invoeren gegevens • Textfields • Radiobuttons • Checkboxes • SelectOne • SelectMany • Select shuttle	Het invoeren van gegevens via de verschillende mogelijke invoer componenten zoals die voor webapplicaties mogelijk zijn. Deze gegevens moeten gevalideerd kunnen worden, en ingevoerde waarden moeten bij het opnieuw tonen van een formulier bewaard blijven.
Data tabellen inclusief sortering en paging	Het weergeven van een lijst met data in tabelvorm, zoals een lijst met gegevens van klanten. Deze data moet gesorteerd en over meerdere pagina's verdeeld kunnen worden.
Data tabel met Master/Detail vervolgpagina	Een zoals hierboven beschreven tabel, waarbij elk element kan worden bewerkt. Om een item te bewerken wordt er een nieuwe pagina geopend.
Data tabel met inline editing	Een lijst met data, waarin de getoonde data direct gewijzigd kan worden zodat meerdere items op een snelle manier tegelijkertijd kunnen worden aangepast.
Uploaden bestanden	Naar een webapplicatie moeten bestanden kunnen worden geupload m.b.v. een file input. Het geuploade bestand moet vervolgens op de server opgeslagen en eventueel bewerkt kunnen worden. Een voorbeeld hiervan is het uploaden van een foto bij een product, waar op de server een thumbnail van wordt gemaakt.
Wizards	Het invoeren van gegevens moet eenvoudig kunnen worden verdeeld over meerdere pagina's.
Dynamische plaatsing	Soms is het noodzakelijk om dynamisch invoer componenten te

input componenten	genereren. Bijvoorbeeld om een lijst met radio buttons te genereren met gegevens uit een database.
Dynamische menu's	Als er dynamische content op een website is te vinden, is er in veel gevallen ook dynamische navigatie nodig naar de content.
Rich Text editor	Gebruikers moeten niet alleen platte tekst kunnen invoeren, maar ook opgemaakte tekst, die wordt opgeslagen in html.

8.2.4 Herbruikbaarheid

Deze lijst met generieke requirements is herbruikbaar voor andere evaluatie trajecten met FPMM, zolang dit gebeurt binnen dezelfde scope van applicatietype.

8.3 Kwalitatieve analyse (stap 3)

8.3.1 Introductie JSF

JSF is een framework gericht op het ontwikkelen van webapplicaties. Het doel van JSF is om het ontwikkelen van een webapplicatie eenvoudiger en sneller te maken. JSF doet dit door een component, en event gebaseerd model aan te bieden. Door standaard componenten te gebruiken kan applicatie functionaliteit eenvoudig geïmplementeerd worden. Het event-model zorgt ervoor dat ontwikkelaars met events kunnen werken zoals dit bij bijvoorbeeld een Swing applicatie gebeurd. Dit in tegenstelling tot het direct werken met een request en response zoals bij de meeste, niet component gebaseerde, frameworks gebeurd. Door een abstractie te vormen bovenop het request/response model kunnen ontwikkelaars zich richten op het implementeren van applicatie functionaliteit in plaats van functionaliteit vertalen naar een request/response model. De belangrijkste punten die JSF biedt zijn als volgt beschreven in de JSF specificatie:

- Het verwerken van de "state" van objecten tussen requests.
- Encapsuleren van de verschillen tussen verschillende browsers en clients.
- Ondersteuning bieden voor het werken met formulieren.
- Het bieden van een sterk getypeerd event-model dat het mogelijk maakt door de client gegenereerde events af te handelen op de server zonder afhankelijk te zijn van HTTP.
- Validatie van request data, en afhandeling van foutmeldingen.
- Conversie tussen data objecten van de applicatie en Strings. Dit omdat request data altijd als String verstuurd wordt.
- Afhandeling van fouten en excepties, en het rapporteren van fouten aan de gebruiker.
- Navigatie tussen pagina's als reactie op events gegenereerd door de UI of model interactie.
-

8.3.2 De plaats van JSF

De meeste webframeworks, waaronder Struts, WebWork en Spring Web-MVC, richten zich vooral op het ondersteunen van het "model 2" model. Model 2 is een implementatie van het Model-View-Controller pattern. Het belangrijkste doel van dit pattern is om de logica van een applicatie los te koppelen van de gebruikers interface. Een request bij een model 2 aanpak wordt als volgt verwerkt:

- De Controller ontvangt de request.
- Afhankelijk van de request wordt er data klaargezet vanuit het Model.
- Afhankelijk van de request kiest de Controller de juiste View.
- De afhandeling van de request wordt doorgegeven aan de View.

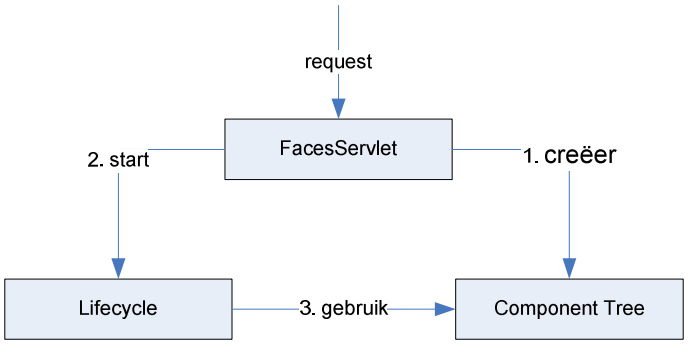
- De View toont de door de Controller klaargezette data.

Hierbij is de Controller doorgaans een Servlet, de View een JSP pagina en het Model de data of services waar de applicatie gebruik van maakt. Het typerende aan deze methode is dat bij elke request een methode op de controller wordt aangeroepen, waar vanuit het verwerken van de request begint. Het afhandelen van events moet hierin handmatig geprogrammeerd worden door request parameters uit te lezen en op basis hiervan de juiste methodes aan te roepen.

JSF maakt dit model, vanuit het oogpunt van een ontwikkelaar, een stuk eenvoudiger. In de View (een JSP pagina) kunnen eenvoudigweg methodes van een normale Java klasse worden aangeroepen. Dit model lijkt veel meer op het model dat gebruikt wordt bij bijvoorbeeld Swing voor het ontwikkelen van desktop applicaties. Met gebruik van dit model hoeft er niet direct met een request gewerkt te worden, het framework abstraheert dit volledig. Het grote verschil is dat bij deze aanpak de View centraal staat en niet de request en controller. Concreet betekent dit dat er minder code zelf hoeft te worden geschreven.

8.3.3 De architectuur van JSF

JSF maakt, net als de meeste webframeworks, gebruik van het Front-Controller design pattern. Bij dit pattern wordt elke request opgevangen door een centrale Servlet. Bij JSF is dit de FacesServlet. Deze Servlet zorgt dat de opgevraagde pagina niet als normale pagina wordt behandeld, maar als JSF pagina. Een van de belangrijkste dingen die deze Servlet doet, is het maken van een Component Tree. Dit is een boomstructuur van een pagina, waarin alle componenten zijn opgenomen. Via deze boomstructuur kunnen componenten die zijn gedefinieerd op de View benaderd worden vanuit de code die achter een pagina werkt. Bij elke request wordt als eerste deze boomstructuur opgebouwd. Als de Component Tree is opgebouwd, wordt de request verder afgehandeld. Dit gebeurd over verschillende fases. Meer detail over de verschillende fases is te vinden in de specificatie van JSF. Het beschreven model is weergeven in Figuur 8-1.



Figuur 8-1 De verschillende fases voor het opbouwen van een JSF pagina

Bij het afhandelen van een request maakt JSF gebruik van een aantal componenten zoals weergegeven in Tabel 8-1. Door het gebruik van renderers, kunnen UI Components flexibel gebruikt worden. Door het ontwikkelen van aparte renderers kan de weergave van een component volledig aangepast worden, of voor een bepaalde taal beschikbaar worden gemaakt (bijvoorbeeld WML i.p.v. HTML).

Component	Omschrijving
UI Component	Een “stateful” , op de server bestaand object. De Component Tree is gevuld met deze componenten.
Renderer	Object dat verantwoordelijk is voor het renderen (weergeven) van

	een UI Component. Een renderer wordt ontwikkeld voor een bepaald UI Component. De standaard renderers geven UI Components weer als HTML.
Validator	Object dat gebruikt wordt om gebruikersinvoer te valideren.
Converter	Object om verstuurde waarden te converteren naar het juiste type. Dit is noodzakelijk omdat request parameters slechts Strings kunnen zijn.
Backing Bean	JavaBeans die kunnen worden gebruikt als onderliggende code voor een pagina. Backing Beans bevatten onder andere event listeners.
Events and listeners	JavaBean event listeners, welk gebruikt kunnen worden om events van de pagina af te handelen.

Tabel 8-1

Dit is uiteraard een erg korte beschrijving van de opbouw van JSF. Een meer uitvoerige beschrijving is niet relevant voor dit onderzoek, en kan worden gevonden in de JSF specificatie [Sun JSF 2004], of in andere beschikbare literatuur.

8.3.4 Ondersteuning JSF

De architectuur van JSF is zodanig opgezet dat een ontwikkelaar zo min mogelijk “boiler-plate” code hoeft te schrijven. Dit betekent dat het framework een grote abstractie biedt bovenop het HTTP protocol en het Servlet framework. Dit is vooral terug te zien in het feit dat ontwikkelaars vrijwel niets te maken hebben met afzonderlijke requests, en bijvoorbeeld het uitlezen van request parameters. Doordat het framework dit afhandelt, wordt dit werk uit handen van ontwikkelaars genomen. Dit zou een grote productiviteitswinst op moeten leveren. Ook de component gebaseerde opzet van JSF biedt in theorie grote voordelen. Componenten zijn configureerbaar, waardoor deze flexibel in te zetten zijn en zodoende herbruikbaar zijn. Dit voordeel wordt groter naarmate er meer aanbieders komen van component bibliotheken. Door het gebruik van kant en klare componenten hoeft er minder code zelf te worden geschreven.

De architectuur van JSF is zeer flexibel. Het framework is “plugable”, wat betekent dat elk onderdeel van het framework eenvoudig kan worden vervangen door eigen implementaties. De mogelijkheden lijken hierdoor onbeperkt. Uiteraard moet hierbij wel worden gekeken naar hoe noodzakelijk zulke aanpassingen zijn. Is dit alleen noodzakelijk in zeer specifieke gevallen, of voldoet de standaard implementatie voor geen enkel systeem.

8.3.5 Theoretische winst

Kijkend naar de ondersteuning en uitbreidingsmogelijkheden die JSF biedt, lijkt het niet meer dan onvermijdelijk dat er met JSF vele malen efficiënter gewerkt kan worden dan met andere, request based, frameworks. Dit vraagt echter wel een andere manier van denken, waarbij een ontwikkelaar rekening moet houden met de process-cycle van het framework. Het is niet ondenkbaar dat dit problemen oplevert bij het ontwikkelen in de praktijk, of dat dit in elk geval zorgt voor een hoge leercurve. De opzet van JSF is absoluut superieur aan dat van vele andere frameworks zoals Struts en Spring WebMVC. De vraag is echter of dit echter niet zoveel complexiteit met zich meebrengt dat de productiviteitwinst hierdoor tegen wordt gehouden, en een eenvoudiger framework waarbij meer handwerk noodzakelijk is misschien wel productiever werkt. Dit zal worden onderzocht in de kwantitatieve analyse, waar de frameworks in de praktijk getest worden.

8.4 Opstellen FO testapplicatie (stap 4)

8.4.1 Opzet

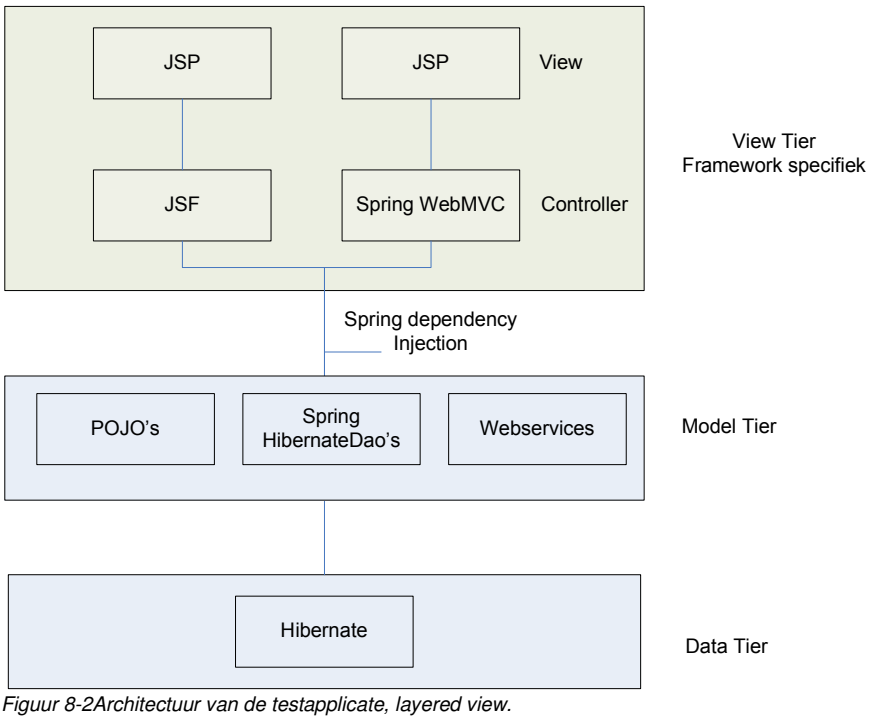
Als testapplicatie is er gekozen voor een simpele webshop applicatie. In deze applicatie kunnen vrijwel alle opgestelde generieke requirements worden geïmplementeerd. De grootte is zodanig dat er meerdere dagen nodig zijn om de applicatie te implementeren. Hierdoor zullen de verschillen tussen JSF en het referentie framework Spring WebMVC duidelijk naar voren komen.

In bijlage A zijn de opgestelde use-cases bijgevoegd. Deze use-cases zijn herbruikbaar voor het evalueren van andere frameworks, zolang dit gebeurt met dezelfde scope van requirements.

8.5 Implementatie niet web gerelateerde code (stap 5)

8.5.1 Architectuur testapplicatie

JSF is een UI framework. Tijdens de test moet er dan ook alleen tijd worden gemeten die besteedt wordt aan het ontwikkelen van dit onderdeel van de applicatie. Om dit mogelijk te maken is de testapplicatie zoveel mogelijk opgedeeld in lagen, waarbij de UI een aparte laag vormt (zoals gebruikelijk is in een n-tier applicatie). Beide implementaties van de testapplicatie maken gebruik van dezelfde code voor de lagen onder de UI laag. Hierin is onder andere toegang naar de database geregeld. Om de verschillende lagen aan elkaar te koppelen wordt het Spring framework gebruikt.



Figuur 8-2Architectuur van de testapplicate, layered view.

8.6 Uitvoeren metingen (stap 6)

8.6.1 Keuze referentie framework

Zoals eerder beschreven heeft het weinig zin om te meten of er met JSF productiever gewerkt kan worden dan zonder het gebruik van een framework. Dit zal namelijk weinig verrassende resultaten opleveren. Wel interessant is het om te kijken of er met JSF productiever gewerkt kan worden dan met een ander framework, wat zijn nut reeds heeft bewezen.

Voor deze casus is het Spring WebMVC framework [Spring][Walls, Breidenbach 2005] gebruikt als referentie. Spring WebMVC is onderdeel van het Spring framework. Spring is een op Dependancy Injection en Aspect Oriented Programming gebaseerd framework, wat helpt bij het ontwikkelen van loosely coupled applicaties. Spring bestaat uit een aantal modules die onafhankelijk van elkaar gebruikt kunnen worden. De WebMVC module is een webframework, dat veel lijkt op Struts.

8.6.2 Gebruik tooling

In een vooronderzoek, te vinden in bijlage B, voorafgaand aan dit onderzoek is vastgesteld dat op dit moment Oracle JDeveloper de beste ondersteuning voor JSF biedt. Deze IDE is tijdens het onderzoek gebruikt voor het implementeren van de test applicatie. Spring WebMVC is onderdeel van het Spring framework. Voor dit framework is er IDE ondersteuning te verkrijgen d.m.v. een plugin voor Eclipse. Deze plugin biedt slechts hulp bij het werken aan de configuratie bestanden van Spring. Omdat dit vrijwel geen winst oplevert bij het werk dat gedaan wordt bij het implementeren van de applicatie met Spring WebMVC, is er gekozen om deze plugin niet te gebruiken, en ook voor de Spring WebMVC implementatie gebruik te maken van JDeveloper. Op deze manier kan exact dezelfde manier van deployment gebruikt worden voor beide implementaties, wat voorkomt dat hier onderlinge verschillen in productiviteit optreden.

8.7 Resultaten en conclusies (stap 7)

8.7.1 Inleiding

In dit hoofdstuk zijn de resultaten en daaraan verbonden conclusies die uit de case studie voort zijn gekomen opgenomen. Als eerste zijn de meetgegevens weergegeven. Per use-case is er een onderverdeling gemaakt tussen de onderdelen waaruit de use-case bestaat. Deze onderverdeling is gemaakt om later een zo goed mogelijk beeld te kunnen geven van de sterke en zwakke punten van JSF. Elke use-case uit de testappplicatie is vertegenwoordigd door een tabel. In deze tabel zijn de verschillende onderdelen weergegeven, waarbij per onderdeel de volgende resultaten zijn opgenomen:

Meting	Uitleg
Ontwikkeltijd JSF	Tijd in minuten
Debugtijd JSF	Tijd in minuten
Regels code JSF	Aantal regels code, gemeten met Clover*
Ontwikkeltijd WebMVC	Tijd in minuten
Debugtijd WebMVC	Tijd in minuten
Regels code WebMVC	Aantal regels code, gemeten met Clover*
Productiviteits factor	Ontwikkeltijd WebMVC / Ontwikkeltijd JSF

* Clover is een plugin voor JDeveloper waarmee metrieken kunnen worden uitgevoerd.

8.7.2 Meetresultaten

Use case: Product beheer (Inline editing)

Onderdeel	Inline editing tabel	Select groups shuttle	Beheer foto's (file upload)	Nieuw product	Totaal
Ontwikkeltijd JSF	106	50	45	20	221
Debugtijd JSF	60	0	0	0	60
Regels code JSF	191	-	-	-	191
Ontwikkeltijd WebMVC	130	90	70	30	320
Debugtijd WebMVC	90	30	0	0	120
Regels code WebMVC	131	61	87	-	279
Productiviteitsfactor	1,23	1,80	1,55	1,50	1,45

Use case: CRUD groepen (Master-detail)

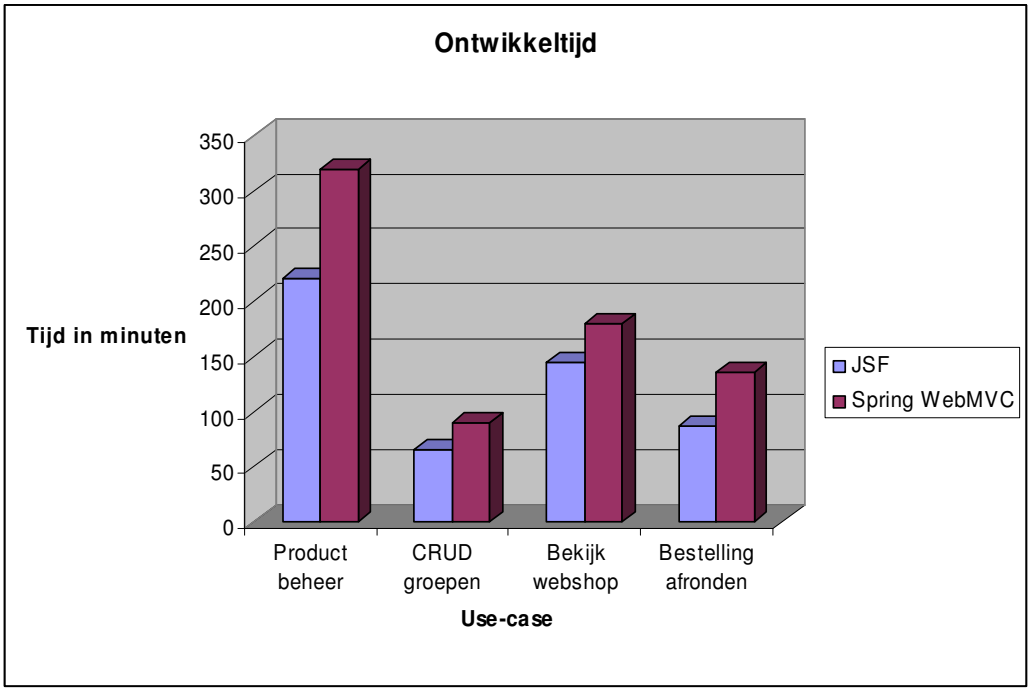
Onderdeel	Overzichts tabel	Edit groep	Nieuwe groep (hergebruik edit)	Verwijder groep	Totaal
Ontwikkeltijd JSF	15	25	15	10	65
Debugtijd JSF	0	10	0	5	15
Regels code JSF	92	-	-	-	92
Ontwikkeltijd WebMVC	20	40	15	15	90
Debugtijd WebMVC	10	20	0	0	30
Regels code WebMVC	19	64	-	24	107
Productiviteitsfactor	1,33	1,6	1	1,5	1,38

Use case: Bekijk webshop

Onderdeel	Groepen menu	Producten overzicht	Product details	Bekijk winkelmandje	Totaal
Ontwikkeltijd JSF	45	20	50	30	145
Debugtijd JSF	5	10	20	8	43
Regels code JSF	-	-	-	-	-
Ontwikkeltijd WebMVC	65	25	40	50	180
Debugtijd WebMVC	20	5	15	15	55
Regels code WebMVC	-	-	-	-	-
Productiviteitsfactor	1.44	1,25	0.8	1,67	1,24

Use case: Bestelling afronden

Onderdeel	Persoonlijke gegevens	Betaalmethode	Overzicht bestelling	Bevestiging	Totaal
Ontwikkeltijd JSF	22	10	45	10	87
Debugtijd JSF	5	0	15	0	20
Regels code JSF	-	-	-	-	-
Ontwikkeltijd WebMVC	40	15	70	10	135
Debugtijd WebMVC	10	5	25	0	40
Regels code WebMVC	-	-	-	-	-
Productiviteitsfactor	1,82	1,5	1,56	1	1,55



Figuur 8-3 Overzicht van ontwikkeltijd

8.7.3 Algemene conclusie

JSF blijkt bij het ontwikkelen van de meeste functionaliteit een flinke productiviteitswinst te bieden ten opzichte van Spring WebMVC. Dit komt vooral door de component gebaseerde manier van werken. Hierdoor hoeft er voor veel standaard functionaliteit, zoals het bewaren van waarden tussen requests en paging in tabellen, vrijwel geen code te worden geschreven. Alle in het functioneel ontwerp gedefinieerde functionaliteit kon met JSF zonder problemen geïmplementeerd worden. Dit wijst er op dat JSF, met gebruik van de ADF Faces bibliotheek, compleet genoeg is voor het ontwikkelen van applicaties binnen de scope die gedefinieerd is in paragraaf 8.2.2.

8.7.4 Sterkte/zwakte analyse

JSF presteert erg goed als het gaat om CRUD systemen. Invoer componenten zorgen zelf voor het bewaren van hun waardes tussen requests, en validatie is bijzonder eenvoudig te implementeren. Bij het gebruik van een bibliotheek als ADF Faces is er de beschikking over een aantal waardevolle componenten. Een goed voorbeeld hiervan is een datatabel waar paging en sorting geïmplementeerd kan worden zonder hiervoor code te hoeven schrijven.
Voor een intranet applicatie zijn dit erg veel gebruikte functionaliteiten en dit weegt dan ook erg zwaar.

Een nadeel van JSF is dat er minder directe controle is over de opmaak. Met een framework zoals Spring WebMVC wordt de opmaak volledig vastgelegd in de JSP pagina's. Met deze aanpak moet er meer JSP/HTML met de hand worden geschreven dan bij het gebruik van JSF componenten, maar hierdoor is er volledige controle over de opmaak. Dit betekent niet dat er geen pagina's met een mooie layout met JSF ontwikkeld kunnen worden, maar wel dat dit meer tijd en moeite kan kosten.

8.7.5 Inschatting leercurve

JSF is een complex framework vergeleken met Spring WebMVC. De reden hiervoor is dat JSF niet controller-centric is, zoals besproken in paragraaf 8.3.3. Dit is in principe een voordeel, want dat is ook de achterliggende reden dat er met JSF erg snel functionaliteit geïmplementeerd kan worden. Het zorgt er echter ook voor dat de afhandeling van een request veel minder eenvoudig te doorgronden is.

Dit zal nieuwe JSF ontwikkelaars in het begin afremmen, en daar moet rekening mee worden gehouden. Een ander probleem voor startende JSF ontwikkelaars is dat een aantal beperking van componenten uit de referentie implementatie niet goed gedocumenteerd zijn. Dit levert moeilijk te verklaren fouten op.

8.7.6 Conclusie gebruik van JSF

Na het gedane onderzoek is er als conclusie getrokken dat JSF een goede keuze is voor projecten die vallen binnen de genoemde scope, die over een langere tijd ontwikkeld zullen worden. Het voordeel van JSF is vooral de productiviteit die gehaald kan worden bij het implementeren van veelgebruikte functionaliteit. De herbruikbaarheid van de code en componenten is echter ook een groot voordeel.

Voor korte projecten die snel moeten worden opgeleverd is JSF wellicht geen goede keuze, aangezien de leercurve vrij steil is. Dit geldt uiteraard alleen voor ontwikkelaars zonder JSF ervaring.

Voor projecten buiten de scope dan behandeld in dit onderzoek, bijvoorbeeld projecten waar een complexe vormgeving wordt gebruikt, zal er een vervolg onderzoek moeten worden gedaan om te bepalen of JSF ook hierbij goed te gebruiken is. Eventueel kan hierbij worden gekozen om een gedeelte van de applicatie te ontwikkelen met JSF en een gedeelte met een ander webframework.

9 Analyse van FPMM

9.1 Inleiding

In dit hoofdstuk wordt er teruggekeken naar het gedane onderzoek, en het resultaat dat hieruit is voortgekomen. Hierbij wordt gekeken of het gestelde doel is bereikt en in hoeverre er nog verder onderzoek noodzakelijk is.

9.2 Terugblik en samenvatting

Terugkijkend op het onderzoek is het verwachte resultaat absoluut gehaald. Met de ontwikkelde methode was het mogelijk om een goed gefundeerd antwoord aan Info Support B.V. te geven wat betreft JSF. Hiermee kan Info Support B.V. ook haar klanten van goed advies voorzien.

De ontwikkelde methode is bruikbaar voor soortgelijke evaluaties en zal in de toekomst opnieuw gebruikt worden voor het evalueren van andere webframeworks.

FPMM is ontwikkeld om onafhankelijk van een individuele ontwikkelaar de productiviteitswinst die een framework biedt ten opzichte van een referentie framework te meten. Om dit mogelijk te maken stelt FPMM de volgende voorwaarden:

- De productiviteit wordt bepaald aan de hand van een vergelijking tussen twee frameworks, en niet op basis van tijdsmetingen bij het gebruik van slechts één framework.
- Het functioneel ontwerp van de te implementeren testapplicatie moet voldoende groot zijn zodat de gemeten productiviteit niet afhankelijk is van hoe een ontwikkelaar zich op een bepaald moment voelt.
- Tijd besteed aan debuggen wordt niet meegeteld als bestede ontwikkeltijd voor de productiviteitsmeting. Dit is noodzakelijk om een eerlijke vergelijking te kunnen maken tussen frameworks waar de ontwikkelaar een verschillende hoeveelheid ervaring mee heeft.

FPMM heeft als doel de productiviteit van een framework in de praktijk te kunnen meten. Om deze reden is FPMM niet gebaseerd op een architectuur onderzoek, maar op een praktijktest. Een architectuur onderzoek zou niet voldoende zijn, aangezien een zeer goede architectuur nog geen garantie is voor een productief framework. Een architectuur analyse is wel onderdeel van FPMM.

9.3 Conclusies met betrekking tot de case studie

9.3.1 Verband regels code en productiviteit

Kijkend naar de meetwaarden in de casestudy, is het opvallend dat bij het gebruik van JSF niet alleen de ontwikkeltijd minder is, maar ook het aantal regels code minder is. Een conclusie die hieruit zou kunnen worden getrokken is dat het aantal regels code blijkbaar wel degelijk directe invloed heeft op de productiviteit. Dit is in tegenstelling tot de stelling dat het aantal regels code niets zegt over productiviteit uit hoofdstuk 6.3.1.

Ondanks dat de cijfers bovenstaande conclusie onderbouwen, is er een betere verklaring voor het verband tussen de productiviteit en het aantal regels code. Bij Spring WebMVC moet er voor elke pagina een aparte controller worden ontwikkeld. Dit is volgens het model van request-based afhandeling. Hierdoor moet er voor elke nieuw te ontwikkelen pagina een stukje infrastructuur worden geregeld. Er moeten een aantal nieuwe bestanden worden aangemaakt, de controller moet worden opgenomen in de configuratie bestanden, en de juiste methoden moeten geïmplementeerd worden. Dit is geen bijzonder ingewikkeld werk, maar kost bedui-

dend meer tijd dan een nieuwe action listener aan een bestaande JSF Backing Bean toevoegen.

Een andere reden voor het verschil in regels code tussen beide frameworks, is dat JSF voor een aantal zaken dusdanige ondersteuning biedt, dat een ontwikkelaar hier vrijwel geen code voor hoeft te schrijven. Een goed voorbeeld hiervan is validatie. Validatie wordt in JSF declaratief op de pagina geregeld, wat inhoudt dat er een extra component op de pagina wordt gezet. In Spring WebMVC moeten hier echter minimaal twee aparte klassen voor worden geschreven, waarin de validatie met de hand moet worden uitgecodeerd, en er is een stukje configuratie noodzakelijk. Dit resulteert in meer werk, maar ook in meer regels code. De winst van JSF zit hierbij echter niet in het absolute aantal regels code dat er minder hoeft te worden geschreven, maar in het feit dat het framework het werk volledig uit handen neemt en de ontwikkelaar hier vrijwel niet meer zelf over na hoeft te denken.

Ondanks dat er een verband lijkt te zijn tussen productiviteit en het aantal regels code, is dit verband minder absoluut dan dat het op het eerste gezicht lijkt. Onderliggend is de reden dat er minder infrastructuur hoeft te worden geregeld, en dat het JSF framework meer werk uit handen neemt.

9.3.2 Het gebruik van Running-Tested-Features

Er is gekozen om in FPMM functionaliteit van de testapplicatie te definiëren als Running-Tested-Features. Een andere mogelijkheid was het gebruik van functiepunten, zoals bij projecten in de praktijk vaak wordt gedaan. Terugkijkend naar de casestudy, is het gebruik van RTF echter erg praktisch gebleken. Door functionaliteit bij naam te noemen, kunnen er per RTF meetwaarden worden opgenomen, zoals gedaan is bij de casestudy. Hierdoor kan er worden geanalyseerd of een framework sterk of zwak is in het implementeren van bepaalde typen functionaliteit. Een goed voorbeeld hiervan is de “product details” pagina uit de casestudy. Hieruit blijkt dat JSF niet zo sterk is in het ontwikkelen van pagina’s met een complexe layout. Als dezelfde metingen waren weergegeven met behulp van functiepunten, zou JSF wel minder sterk naar voren komen, maar het zou niet duidelijk worden dat JSF alleen slecht presteert bij dit type pagina. Deze informatie is cruciaal bij het kiezen van een geschikt framework voor een project, en daarom is het gebruik van RTF beter dan het gebruik van functiepunten.

9.3.3 Nulmetingen als validatie van FPMM

Tijdens de casestudy zijn er een aantal nulmetingen gedaan. Dat wil zeggen dat er met JSF en SpringWebMVC exact evenveel tijd nodig was om een Running-Tested-Feature (RTF) te implementeren. Kijkend naar de bijbehorende beschreven functionaliteit blijkt dit een stukje validatie te leveren voor FPMM. Bij de RTF’s waarbij de nulmetingen werden gedaan, blijkt namelijk dat beide frameworks hun kracht niet konden gebruiken voor dat type functionaliteit. Dit betekent dat de functionaliteit vrijwel zonder extra hulp van het framework is geïmplementeerd. Als hierbij blijkt dat bij beide frameworks dezelfde tijd wordt gemeten, is dit een bewijs dat de gemeten tijd niet wordt beïnvloed door externe factoren die los staan van het framework.

9.3.4 Afwijkende meetwaarden

In de casestudy naar JSF zijn de resultaten erg gelijkgestemd, er zijn weinig uitschieters. Bovendien scoort JSF op vrijwel alle onderdelen 20-40 % beter dan Spring WebMVC. De vraag is of deze beperkte bandbreedte komt door de opzet van FPMM, of dat dit typerend is voor JSF.

Het antwoord op deze vraag is tweeledig. Hierbij is er een duidelijk onderscheid tussen meetwaarden van het implementeren van soortgelijke functionaliteit, en het implementeren van heel verschillende functionaliteit.

Door de opzet van een framework, wordt het implementeren van bepaalde functionaliteit makkelijker of moeilijker. Een voorbeeld hiervan is het state-management model van JSF, waardoor het implementeren van CRUD functionaliteit erg eenvoudig is. Voor andere typen functionaliteit biedt een framework misschien minder goede mogelijkheden, waardoor er meer handwerk gedaan moet worden en hier op productiviteit wordt ingeleverd. Bij meetwaarden van verschillende typen functionaliteit is het daarom heel goed mogelijk dat hier zeer afwijkende resultaten naar voren komen. Dit geeft dan aan dat het framework op bepaalde punten heel sterk is, en op andere punten zwak. Hier kan dan rekening mee worden gehouden bij het selecteren van een framework voor een project.

Het feit dat bij de casestudy de resultaten een vrij beperkte bandbreedte hebben, zegt dan ook iets over JSF. Namelijk dat het framework blijkbaar erg compleet is, en op alle geteste vlakken goede ondersteuning biedt. Ook zegt het iets over het referentie framework. Aangezien JSF ook geen extreem positieve uitschieters heeft, biedt het referentie framework blijkbaar ook voor alle geteste onderdelen goede ondersteuning.

Bij soortgelijke functionaliteit mogen de resultaten niet teveel afwijken. Dit zou namelijk een afwijking zijn die niet door het framework, maar door externe factoren wordt veroorzaakt. FPMM is zo opgezet dat dit tot een minimum wordt beperkt. De casestudy toont aan dat dit inderdaad zo is, doordat er een duidelijke lijn is te vinden in de productiviteit bij het implementeren van soortgelijke functionaliteit, en door de nulmetingen zoals beschreven in paragraaf 9.3.3.

9.3.5 Ervaring en debuggen

Bij FPMM wordt tijd besteed aan debuggen niet meegeteld als ontwikkeltijd, omdat de hoeveelheid debugtijd sterk afhankelijk is van de algemene ervaring van een ontwikkelaar, en de ervaring met een specifiek framework. Als we echter kijken naar de casestudy, blijkt dat er bij een aantal onderdelen met JSF minder tijd aan debuggen is besteed dan met Spring WebMVC. Dit is vreemd, aangezien er tijdens het uitvoeren van de casestudy veel meer ervaring met Spring WebMVC aanwezig was dan met JSF. Betekent dit dat ervaring geen invloed heeft op debuggen?

De verklaring ligt echter op het technische vlak. Met JSF zijn een aantal zaken zoveel eenvoudiger gemaakt dat hier bijna geen fouten mee gemaakt kunnen worden. Ondanks dat er voldoende Spring WebMVC ervaring aanwezig was, was de implementatie zoveel complexer dat hier toch nog meer fouten mee werden gemaakt.

9.4 Validatie methode

9.4.1 Motivatie

FPMM is zodanig ontwikkeld dat de resultaten onafhankelijk zijn van de persoon door wie de methode gebruikt wordt, zoals beschreven in hoofdstuk 7 .

Tijdens het uitvoeren van de case studie was het echter niet mogelijk om met meerdere ontwikkelaars te werken. Hierdoor is het niet mogelijk geweest te bewijzen dat FPMM inderdaad onafhankelijk van individuele ontwikkelaars dezelfde conclusies oplevert.

Om verder te kunnen bewijzen dat FPMM inderdaad persoonsonafhankelijke resultaten oplevert, zal de methode moeten worden gebruikt in een grootschaliger onderzoek dan de

besproken case studie. Door de methode te gebruiken in een onderzoek waarbij er een groter aantal ontwikkelaars met verschillend hoeveelheid ervaring meewerken, kan worden gevalideerd of de methode inderdaad onafhankelijk van het niveau van de ontwikkelaars dezelfde resultaten oplevert. Als blijkt dat hier onverhoopt toch verschillen in blijken te zijn, kan worden onderzocht waar deze verschillen door veroorzaakt worden, en kan de methode worden aangescherpt.

9.4.2 Uitvoering

In deze paragraaf is beschreven hoe de methode gevalideerd kan worden in een groter onderzoek.

Om te valideren of de methode werkt voor verschillende niveaus van ontwikkelaars, zal er een selectie ontwikkelaars aan het onderzoek moeten meewerken waarvan de ervaring duidelijk verschilt. Dit zal uitwijzen of de methode bij verschillende ervaring niveaus dezelfde uitkomst oplevert.
Voorafgaand aan het onderzoek moeten de deelnemers in ieder geval basiservaring met het te onderzoeken framework hebben, dit is ook één van de stappen van FPMM. Ook moeten de richtlijnen voor het vastleggen van tijd duidelijk zijn bij de deelnemers. Beter is het nog om als onderzoeker zelf de tijd voor elke deelnemer te meten. Dit elimineert het probleem dat deelnemers mogelijk een andere interpretatie van debuggen hebben, en de bestede tijd hieraan verschillend meten.

De webapplicatie die ontwikkeld is voor de case studie is goed te gebruiken als testapplicatie voor dit onderzoek. Deze applicatie is voldoende groot is om representatief te zijn voor een echte applicatie. Belangrijk is ook dat de architectuur van de applicatie zodanig is dat de framework specifieke code duidelijk geïsoleerd kan worden van code die bedoeld is voor het implementeren van bijvoorbeeld business logica en database toegang.

9.5 Beslissingen tijdens het onderzoek

De moeilijkste stap tijdens het project was de beslissing om de evaluatie methode niet te baseren op een architectuur onderzoek. Dit heeft het onderzoek veel complexer gemaakt, omdat hiermee het meten van productiviteit werd geïntroduceerd. Dit is een onderwerp met veel onzekere factoren, wat het moeilijk maakt om hiermee te werken. Achteraf gezien is het wel een goede keuze geweest om productiviteitsmetingen te doen, en zodoende de praktijk te betrekken bij de evaluatie van een webframework. Hierdoor is de methode bruikbaarder voor een praktijk gerichte evaluatie zoals Info Support B.V. nodig heeft.

9.6 Discussie

9.6.1 Debuggen

In FPMM wordt debug tijd niet meegeteld als ontwikkeltijd. Deze keuze is gemaakt om het zodoende mogelijk te maken een vergelijking tussen frameworks te maken, zonder dat dit afhankelijk is van de hoeveelheid ervaring met een specifiek framework.

Een punt van discussie is echter dat debuggen altijd onderdeel van het ontwikkelproces zal zijn, onafhankelijk van hoe goed een ontwikkelaar is. De vraag is dan ook of deze tijd zomaar van de productiviteit cijfers afgetrokken kan worden.

Als debugtijd wel meegenomen wordt in de metingen, wordt er hierdoor een sterke afhankelijkheid gecreëerd met de ervaring van een ontwikkelaar. Vanuit de ervaring van Info Support B.V. en mijn eigen ervaring blijkt dat een meer ervaren ontwikkelaar over het algemeen minder fouten maakt dan een minder ervaren ontwikkelaar. Hetzelfde geldt voor ervaring met een

specifieke techniek zoals een framework. Een eerlijke vergelijking tussen een nieuw framework en een referentie framework waar al jaren mee wordt gewerkt zou daarmee onmogelijk worden.

Om deze reden is er gekozen om debuggen los te zien van ontwikkeltijd. Ideaal is het misschien niet, maar wel de enige realistische mogelijkheid.
Er zijn situaties denkbaar waarin de debugtijd wel als ontwikkeltijd meegeteld kan worden. Een realistisch voorbeeld hiervan is als er voor een nieuw project een keuze moet worden gemaakt tussen twee frameworks. Voor dat project moet het meest productieve framework worden gekozen voor dat moment.

In dit geval is het onbelangrijk of de oorzaak van de productiviteit ligt bij de ervaring van de ontwikkelaars. Het enige antwoord dat belangrijk is, is dat met de op dat moment aanwezige ervaring er met het ene framework productiever kan worden gewerkt dan met het andere framework. In dit geval kan debugtijd worden meegenomen in de productiviteitsmeting.

9.6.2 Procentueel verschil

In FPMM wordt de aanname gedaan dat een ontwikkelaar met meer ervaring minder tijd kwijt is aan het implementeren van de test applicatie dan een ontwikkelaar met minder ervaring. Het procentuele verschil tussen de implementatie met het te evalueren framework en het referentie framework blijft voor beide ontwikkelaars echter hetzelfde. De vraag is of deze aanname wel correct is.

De aanname is gedaan op basis van ervaring van Info Support B.V. Vanuit de ervaring in verschillende projectteams is gebleken dat bij het gebruik van bepaalde techniek, de productiviteit van het hele team verbetert en niet alleen de productiviteit van bijvoorbeeld zeer ervaren ontwikkelaars. Dit bewijst nog niet of de winst daarmee voor elke ontwikkelaar exact even groot is, maar geeft wel aan dat het effect globaal voor alle ontwikkelaars hetzelfde is.

Bij het uitvoeren van FPMM met een grotere populatie ontwikkelaars, kan deze aanname worden bewezen. Tijdens het uitgevoerde onderzoek was dit echter niet mogelijk. Als blijkt dat de aanname onjuist is, zal er in FPMM moeten worden meegenomen voor welk niveau van ervaring de resultaten gelden. Binnen een bedrijf zijn de resultaten daarmee nog steeds prima te gebruiken, aangezien hier meestal goed duidelijk is op welk niveau een bepaalde ontwikkelaar zit.

10 Literatuur

10.1 Boeken

[Beck 2005] Beck K,. Extreme Programming Explained second edition, 2005, Addison-Wesley, 189 pages

[Bergsten 2004] Bergsten H., JavaServer Faces, 2004, O'Reilly, 589 pages

[Hall 2001] Hall M., More Servlets and JavaServer Pages, 2001, 752 pages

[Johnson 2002] Johnson R., “Expert one-on-one: J2EE Design and Development”, 2002, Wrox, 768 pages

[Mann 2005] Mann K.D., JavaServer Faces in Action, 2005, Manning, 702 pages

[McConnell 2004] McConnell S. C., Code Complete second edition, 2004, MS Press, 914 pages

[Walls, Breidenbach 2005] Walls C., Breidenbach R., Spring in Action, 2005, Manning, 472 pages

10.2 Papers

[Bergsten 2004] Bergsten H., Improving JSF by Dumping JSP, 2004, OnJava.com, O'Reilly

[Bonvanie 2002] Bonvanie R., Java Frameworks Take Hold, 2002, Oracle

[CASEMaker] CASEMaker Inc., What is Rapid Application Development?, 32 pages

[Cioroianu 2005] Cioroianu A., Reusability in Web Applications, 2005, Oracle

[DeMarco, Lister 1985] DeMarco, T., Lister T., “Programmer Performance and the Effects of the Workplace.”, IEEE, 1985

[Fayad, Schmidt 1997] Fayad M., Schmidt D.C., Object-Oriented Application Frameworks, 1997, ACM

[JISC] Plans Casal D., Advanced Software Development for Web Applications, JISC, 23 pages

[Johnson, Foote 1988] Johnson R., Foote B., Designing Reusable Classes, 1988, Department of Computer Science, University of Illinois, Urbana-Champaign, Journal of Object-Oriented Programming

[McConnell 2002] McConnell S. C., Quantifying Soft Factors, 2002

[Raible JDJ 2005] Raible M., Challenges in the J2EE Web Tier While Frameworks Driving Innovation, 2005, JDJ, SYS-CON Media Inc.

[Raible VS 2005] Raible M., Java Web Frameworks, 2005, Virtuas Solutions

[Prechelt 1999] Prechelt L., “The 28:1 Grant/Sackman legend is misleading, or: How large is interpersonal variation really?” Universität Karlsruhe, 1999

[Sackman 1968] Sackman H., Erikson W.J., Grant E.E, Exploratory Experimental Studies Comparing Online and Offline Programming Performance, ACM, 1968

[Sun JSF 2004] McClanahan C., Burns E., Kitain R., Java Server Faces Specification version 1.1, Sun Microsystems, 2004, 301 pages

[Verhoef 2005] Verhoef C., Functiepunten: regie of religie, Automatisering Gids, week 42, 2005.

[Vohra 2005] Vohra D., Building a Tiles Application with JSF Components in JDeveloper, 2005, Oracle

[Westkämper 2004] Westkämper T., Architectural models of J2EE Web tier frameworks, 2004, University of Tampere, 67 pages

[Yang Shen 2004] Yang Shen D., Put JSF to work, 2004, JavaWorld

10.3 Online artikelen

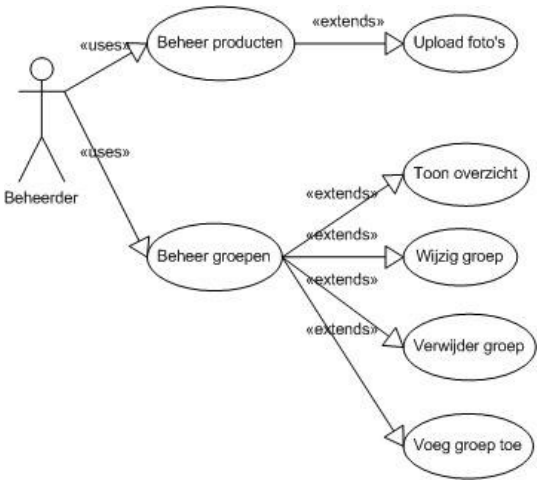
[Fowler 2003] Fowler M., http://www.martinfowler.com/bliki/CannotMeasureProductivity.html, 2003

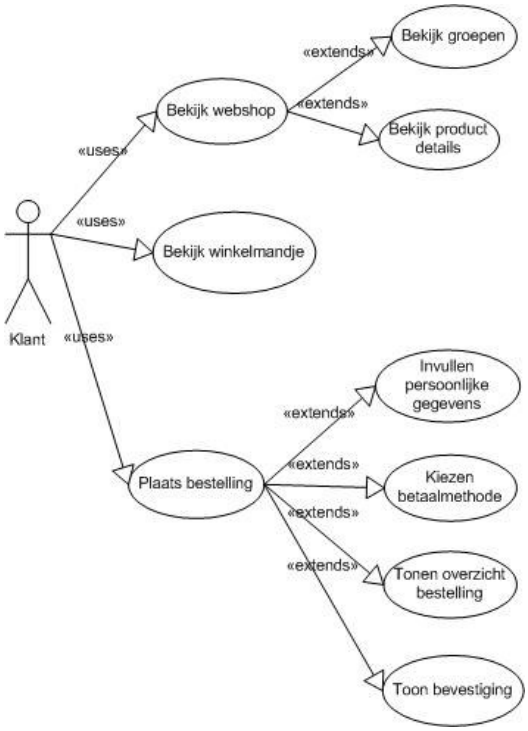
[Fowler 2004] Fowler M., http://www.martinfowler.com/articles/injection.html, 2004

[Raible blog] Raible M., **http://raibledesigns.com/page/rd?anchor=my_jsf_experience**

[Spring] http://www.springframework.org

11 Bijlage A – Use-cases testapplicatie





Naam	Beheer producten
Actors	Beheerder
Omschrijving	Er wordt een lijst met producten weergegeven. Van elk product worden alle gegevens weergegeven. De gegevens moeten kunnen worden gesorteerd en worden verdeeld over meerdere pagina's. In het product overzicht kunnen de getoonde gegevens direct gewijzigd worden (inline editing), om bijvoorbeeld van alle producten in een groep de prijs te verlagen. Deze wijzigingen kunnen in een keer worden opgeslagen.
Aannames	De gebruiker is ingelogd

Naam	Upload / verwijder product foto's
Actors	Beheerder
Omschrijving	De foto's van een product moeten kunnen worden verwijderd, en er moeten nieuwe foto's kunnen worden toegevoegd. Dit gebeurd in een apart scherm waar de foto's in een lijst worden weergegeven.
Aannames	De gebruiker is ingelogd
Alternatieven	

Naam	Beheer groepen
Actors	Beheerder
Omschrijving	Groepen worden beheerd volgens een master-detail opzet. Er wordt een lijst getoond van groepen, waarna er een groep kan worden bewerkt in een apart scherm. Sortering en paging is bij het groepen overzicht niet noodzakelijk. Het scherm voor het toevoegen en wijzigen van groepen wordt gevalideerd op volledigheid van gegevens.
Aannames	De gebruiker is ingelogd
Alternatieven	

Naam	Bekijk webshop
Actors	Klant
Omschrijving	Gebruikers kunnen groepen, en producten in groepen bekijken. Van elk product kan er een detail pagina worden opgevraagd waar ook de foto's staan weergegeven. Vanaf deze pagina kan een product in het winkelmandje worden geplaatst. De in het winkelmandje geplaatste producten kunnen worden weergegeven, het aantal per product kan worden gewijzigd, en producten kunnen uit het mandje verwijderd worden.
Aannames	
Alternatieven	

Naam	Bekijk winkelmandje
Actors	Klant
Omschrijving	Een overzicht van de producten die de klant in het winkelmandje heeft geplaatst. Vanaf deze pagina kan het aantal van een product worden aangepast, en kunnen producten uit het mandje verwijderd worden.
Aannames	
Alternatieven	

Naam	Plaats bestelling
Actors	Klant
Omschrijving	Om een bestelling te plaatsen moet de klant een wizard doorlopen, waarin • persoonlijke gegevens worden opgenomen • een betaalmethode wordt gekozen • er een overzicht van bestelde producten wordt getoond • een bevestiging wordt getoond
Aannames	Aanname A
Alternatieven	

12 Bijlage B – JSF implementaties en tooling

12.1 IBM implementatie

12.1.1 IBM's JSF implementatie

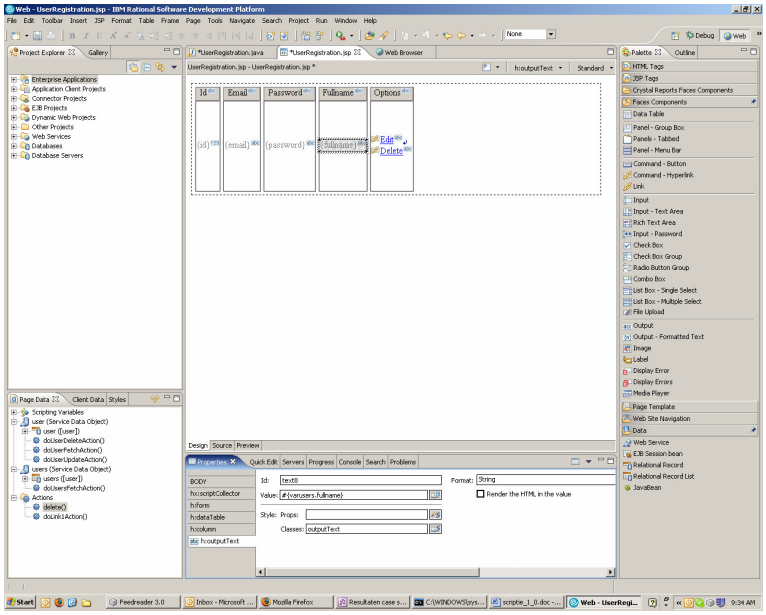
IBM heeft een eigen implementatie van de JSF specificatie. Deze implementatie is gebaseerd op versie 1.0 van de specificatie, en is aangevuld met een aantal bugfixes. De meeste van deze bugfixes maken echter ook deel uit van versie 1.1 van de specificatie. Het kan dan ook als groot nadeel worden gezien dat IBM een eigen JSF implementatie gebruikt, en niet de referentie implementatie zoals bijvoorbeeld Oracle wel doet. Bij nieuwe versies van de JSF specificatie zal de implementatie van IBM mogelijk gaan achterlopen. Ook kan het gebruik van de IBM implementatie problemen opleveren bij gebruik op een andere applicatie/web servers dan die van IBM zelf.

12.1.2 IBM componenten

IBM biedt een kleine aanvulling op de componenten uit de specificatie in de "HTML Extended" bibliotheek. Deze bibliotheek breidt een aantal componenten uit om deze beter configureerbaar te maken en biedt een aantal aanvullende componenten. De uitgebreidere configuratie is vooral bruikbaar om het uiterlijk van componenten aan te passen. De bibliotheek is echter vrij beperkt en waarschijnlijk zal de behoefte aan extra componenten blijven bestaan.

12.1.3 IDE ondersteuning

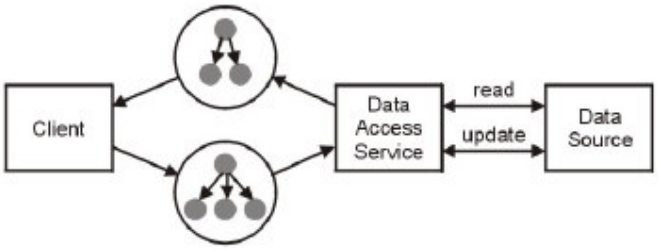
IBM biedt ondersteuning voor JSF met de Rational Application Developer 6.0 IDE. Voor deze IDE moet betaald worden en is gebaseerd op Eclipse. JSF ondersteuning is vooral terug te vinden in de verschillende wizards die de IDE biedt om pagina's te creëren en de grafische editor om componenten op een pagina te slepen. De grafische editor is bruikbaar, maar in de praktijk zal er toch vaak direct aan de JSP code worden gewerkt. De IDE biedt geen code-completion in JSP pagina's wat een duidelijk nadeel is ten opzichte van de Oracle tooling.



Figuur 12-1 Rational Application Developer 6, visueel ontwerpen van JSF pagina's.

12.1.4 IBM SDO

Service Data Objects (SDO) is een framework om eenvoudig met verschillende typen gegevensbronnen te kunnen werken zoals relationele databases, webservices en EJB's. De SDO specificatie is opgezet om op een uniforme manier met deze gegevensbronnen te kunnen werken door een abstractielaag te introduceren tussen een gegevensbron en een applicatie. SDO is ontwikkeld door IBM en BEA, en is sinds 2003 een JSR (JSR 235). Sindsdien is de JSR echter niet meer bijgehouden, en is de SDO specificatie buiten de JSR doorontwikkeld.



Figuur 12-2 De disconnected data architectuur zoals weergegeven in de SDO specificatie

Met behulp van RAD 6.0 kunnen SDO objecten direct gebruikt worden op JSF pagina's. Door een SDO object op een pagina te slepen kunnen er bijvoorbeeld tabellen worden gegenereerd. Uiteraard is het niet noodzakelijk om SDO te gebruiken in een JSF project. Dit wordt echter wel door IBM gepushed.

De vraag is of het op dit moment een goede keuze is om SDO te adopteren in projecten, aangezien het onduidelijk is waar de toekomst van SDO ligt.

12.2 Oracle Implementatie

12.2.1 Oracle ADF Faces

ADF Faces is een door Oracle ontwikkelde verzameling JSF componenten. Voor een deel zijn dit componenten die de standaard componenten vervangen, en voor een deel zijn dit aanvullende componenten. De aanvullende componenten vullen de referentie implementatie aan met zeer bruikbare componenten zoals een menu, tree, date picker en file-upload component. De ADF Faces componenten die componenten uit de referentie implementatie vervangen hebben hetzelfde doel als de standaard componenten, maar geven meer mogelijkheden tot configuratie en bieden bijvoorbeeld ondersteuning voor client-side validatie en partial-postbacks.

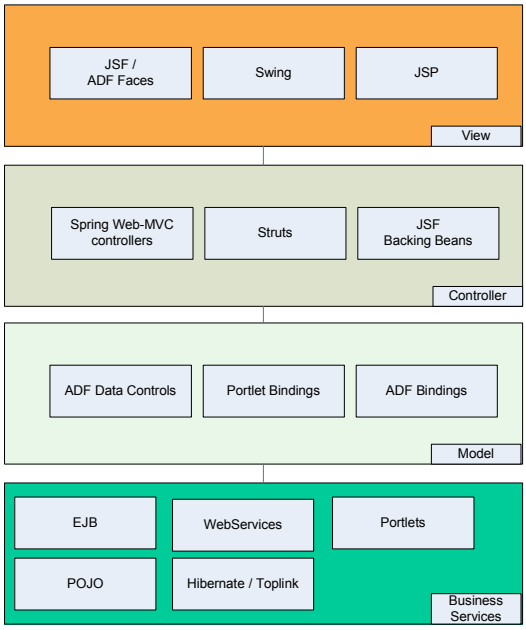
ADF Faces maakt gebruik van de referentie implementatie (versie 1.1) van JSF. Oracle heeft geen eigen JSF implementatie. Dit heeft als groot voordeel dat ADF Faces op elke webserver gebruikt kan worden die compatible is met de referentie implementatie.

12.2.2 Licentie en ondersteuning

Oracle heeft ADF Faces gedoneerd aan het Apache MyFaces project onder de Apache 2.0 licentie. Dit betekent dat ADF Faces zonder licentiekosten te gebruiken is. Binnen het MyFaces project zijn de gedoneerde componenten opgenomen onder subproject Trinidad. De reden dat Oracle ADF Faces aan het MyFaces project gedoneerd heeft is dat Oracle wil proberen JSF toegankelijk te maken, zodat JSF goed kan concurreren met bijvoorbeeld ASP.NET. ADF Faces wordt ondersteund door elke implementatie die de 1.1 specificatie implementeert. Dit betekent dat ADF Faces niet afhankelijk is van een bepaalde applicatie server. Tijdens het onderzoek is dit getest door ADF Faces te gebruiken in een op Tomcat 5.5 draaiende applicatie. In hoeverre ADF Faces door IDE's wordt ondersteund, hangt voor het grootste deel af van de IDE. Oracle biedt ondersteuning voor hen eigen IDE JDeveloper. Als er ook gebruik wordt gemaakt van ADF zijn hier echter wel kosten aan verbonden. Dit staat echter los van JSF.

12.2.3 ADF en architectuur

ADF staat voor Application Development Framework. Dit framework heeft als doel om het ontwikkelen van J2EE applicaties te vereenvoudigen. De architectuur van ADF bestaat uit vier lagen; View, Controller, Model, Business Services. In elke laag kunnen ontwikkelaars kiezen tussen verschillende technologieën. Deze architectuur is weergegeven in onderstaande afbeelding. ADF verbindt de verschillende lagen met elkaar.



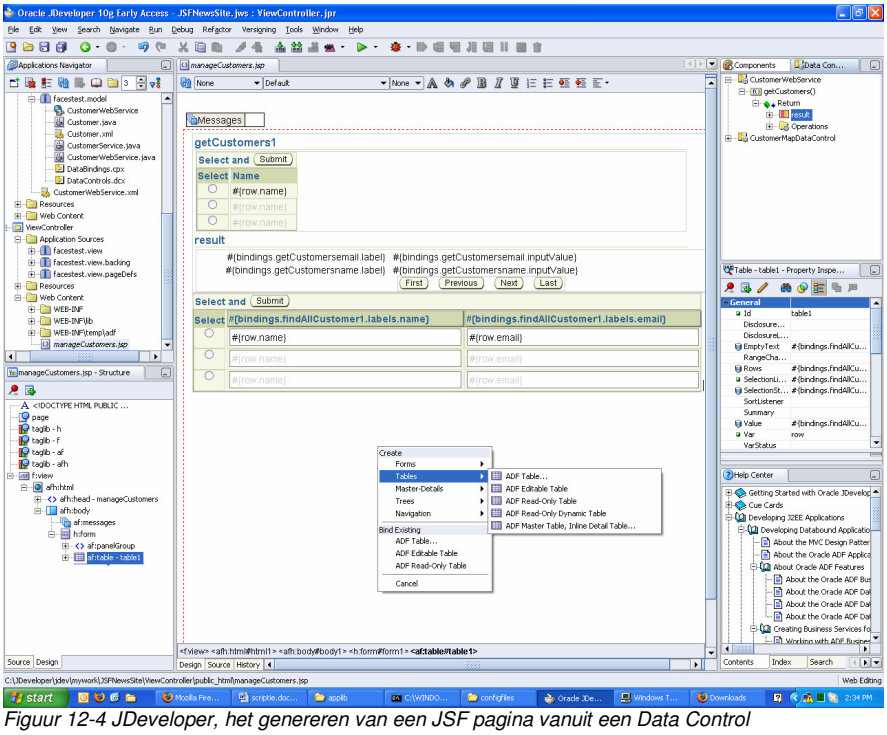
Figuur 12-3 De architectuur van ADF

Zoals te zien is in bovenstaande afbeelding is ADF Faces een onderdeel van ADF.
Of het gebruik van ADF wel of niet verstandig is valt buiten de scope van dit rapport, het is echter wel belangrijk om te kijken naar in hoeverre ADF Faces hier afhankelijk van is. Om ADF Faces gebruik te maken is het niet noodzakelijk om ADF te gebruiken. Dit is een erg belangrijk punt, aangezien ADF Faces op deze manier ingezet kan worden zonder dat dit grote impact op de architectuur van een systeem heeft. Als er wel ADF gebruikt wordt in de architectuur van een systeem, is het wel bijzonder eenvoudig om ADF Faces als view in te zetten. Vanuit de JDeveloper omgeving kunnen er vanuit ADF business componenten eenvoudig JSF pagina's gegenereerd worden die gebruik maken van ADF Faces componenten.

12.2.4 Samenwerking met ADF

Ondanks dat de vraag of het gebruik van ADF wel of niet een goede keuze is in een architectuur buiten de scope van dit rapport ligt, is het wel interessant om te kijken naar welke voordelen de samenwerking tussen ADF en ADF Faces heeft. Een van de belangrijkste onderdelen van ADF is databinding. Oracle probeert databinding binnen het Java Community Process (JCP) te standaardiseren. Databinding koppelt de verschillende lagen van een systeem aan elkaar. Dit wordt gedaan door het maken van een Data Control. Een Data Control is een adapter voor bijvoorbeeld een EJB, Webservice of JavaBean. Door het gebruik van deze adapters wordt er een abstractie laag gecreëerd bovenop verschillende typen services en databases. Hierdoor maakt het voor een client niet uit of er gebruik wordt gemaakt van bijvoorbeeld een WebService of een EJB. Data Bindings worden declaratief geconfigureerd in XML bestanden. Deze XML bestanden worden door de IDE gegenereerd, en hoeven in principe niet met de hand aangepast te worden. Het gebruik van ADF zit wel op een dusdanige manier in de IDE verwerkt, dat het lastig lijkt om ADF te gebruiken zonder JDeveloper.

Nadat er Data Controls gemaakt zijn kunnen deze vervolgens gebruikt worden bij het ontwikkelen van een JSF applicatie. Door in JDeveloper simpelweg een Data Control op een JSF pagina te slepen, kunnen er complexe tabellen en formulieren gegenereerd worden. Als er Data Controls gebruikt worden kan dit enorm in ontwikkeltijd schelen. De gegenereerde code maakt gebruik van ADF Faces componenten. Overigens is deze manier van werken niet alleen mogelijk met JSF, maar ook met Swing.



Figuur 12-4 JDeveloper, het genereren van een JSF pagina vanuit een Data Control

12.2.5 IDE ondersteuning

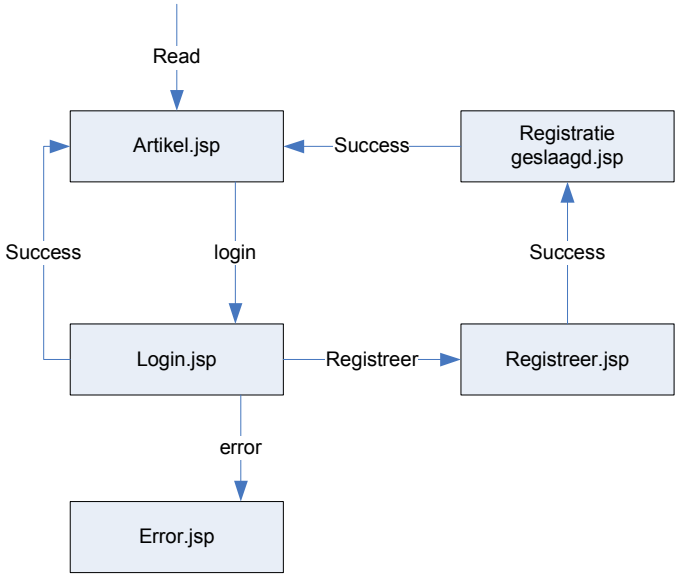
ADF Faces wordt ondersteund door Oracle's eigen ontwikkelomgeving JDeveloper. De JSF ondersteuning hierin is erg uitgebreid. Het belangrijkste punt hierbij is de visuele editor, waarin componenten op een JSF pagina gesleept kunnen worden. Hierin is uiteraard ook de ondersteuning voor ADF Data Controls opgenomen. Ook is er visuele ondersteuning voor het maken van navigation rules en code-completetion voor de JSF Expression Language.

12.2.6 Skins

ADF Faces biedt de mogelijkheid om skins te definiëren. Een skin is een verzameling stylesheets en plaatjes die samen het uiterlijk van een applicatie bepalen. Skins maken het mogelijk het uiterlijk van een applicatie op een centraal punt te configureren, in plaats dat elke pagina apart opgemaakt moet worden. Dit maakt het mogelijk het uiterlijk van een applicatie eenvoudig te wijzigen. De stylesheets bestaan uit CSS 3.0 style declaraties, onderverdeelt in verschillende niveaus om globaal of op component niveau het uiterlijk van een applicatie te definiëren. Skins kunnen erg belangrijk zijn bij de keuze voor ADF Faces. Omdat alle ADF Faces componenten skins ondersteunen, kan het werk van ontwikkelaars en designers beter gescheiden worden. Voor een groot deel zullen designers zich kunnen beperken tot het maken van Skins om de applicatie vorm te geven. Dit pakt ook direct het probleem aan dat designers te beperkt zouden zijn als er zo min mogelijk HTML binnen JSF pagina's gebruikt wordt.

12.2.7 Process / Dialog framework

ADF Faces biedt functionaliteit voor het inrichten van processen binnen een applicatie. Dit kan bijvoorbeeld gebruikt worden om de volgende pageflow te implementeren.



Figuur 12-5 Voorbeeld pageflow bij het gebruik van het Process framework

Voordat een artikel getoond wordt, moet een gebruiker eerst inloggen, en zonodig eerst registreren. Als dit gebeurd is moet de oorspronkelijk opgevraagde pagina getoond worden. Het inloggen en registreren kan hierbij als een proces gezien worden. De Process en Dialog frameworks helpen bij het implementeren van zulke processen. Als een proces beëindigd is zorgt het framework ervoor dat de oorspronkelijk opgevraagde pagina getoond wordt. Met behulp van het Dialog framework kan een proces in een popup venster getoond worden, in plaats dat de huidige pagina vervangen wordt.

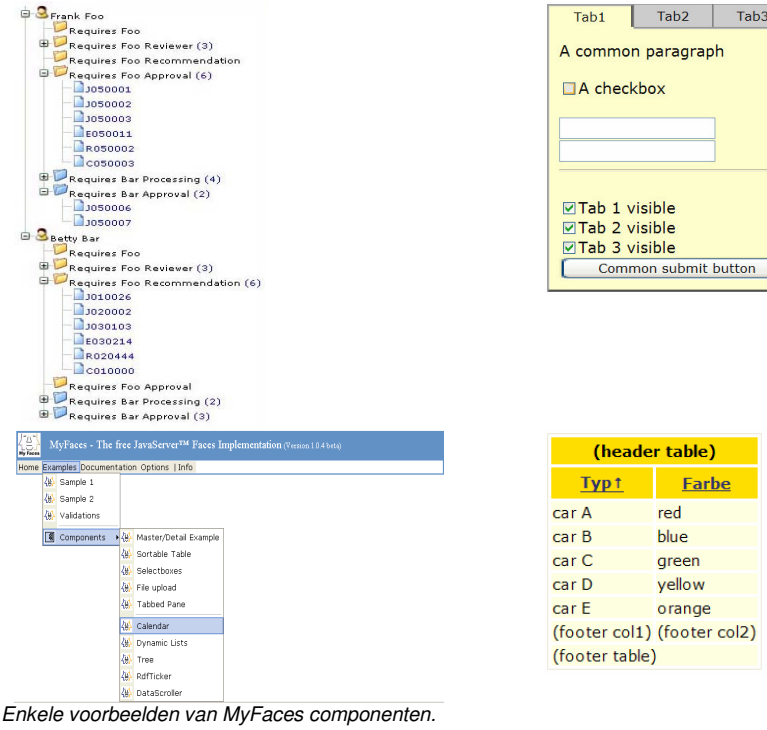
12.3 MyFaces

12.3.1 MyFaces implementatie

MyFaces is een Apache project. Onderdeel van dit project is een implementatie van de JSF specificatie. Deze implementatie is volledig compatible met de specificatie, en biedt op sommige punten aanvullende functionaliteit. Het nadeel van het gebruik van deze implementatie is dat MyFaces niet direct compatible zal worden gemaakt met versie 1.2 van de specificatie. MyFaces wordt door sommigen aangeraden omdat het minder bugs zou bevatten dan de referentie implementatie.

12.3.2 MyFaces componenten

Het meest interessant aan het MyFaces project is de component bibliotheek. Deze bibliotheek is ondergebracht in een subproject genaamd Tomahawk. In de Tomahawk zitten erg praktische componenten zoals een Menu, Tree en Tablayout. Bij een aantal van deze componenten is ook AJAX functionaliteit ingebouwd.



Enkele voorbeelden van MyFaces componenten.

Om de componenten te kunnen gebruiken is het niet noodzakelijk om de MyFaces JSF implementatie te gebruiken. Buiten de componenten van het Tomahawk project, zijn de door Oracle gedoneerde ADF Faces componenten opgenomen in subproject Trinidad. Er is enigszins sprake van overlap tussen de Tomahawk componenten en de Trinidad componenten. In hoeverre hier wel of niet een samenvoeging zal plaatsvinden is op dit moment nog niet duidelijk. Ook is het niet duidelijk of het Trinidad project door Oracle zal worden onderhouden.

12.3.3 MyFaces community

Langzamerhand lijkt MyFaces langzaam uit te groeien tot een defacto standaard. Een aantal grote spelers zoals JBoss bieden bijvoorbeeld JSF ondersteuning in de vorm van MyFaces bij applicatieservers en ontwikkel frameworks. Op dit moment is het nog te vroeg om te zeggen dat JSF een defacto standaard is, maar het is niet uitgesloten dat dit gaat gebeuren.