Adviezen ter realisatie van een professionele technische infrastructuur voor webdevelopmentbureaus

Mark M. Mulder (2009) – ikbenbitterzoet.com
Afstudeerproject Interactieve Media aan de Hogeschool van Amsterdam

Voorwoord

Als eerste wil ik Fullmoon Interactive Solutions te Amsterdam bedanken dat ze zo vrijdenkend zijn geweest om mij uit te horen over hun situatie. Het is niet makkelijk om iemand te horen praten over zaken die mogelijk niet goed gaan bij het bedrijf, maar ze waren juist erg blij dat ik sommige dingen heb aangekaart. Daarnaast kreeg ik veel ruimte en hebben ze goed meegedacht. Ik ben blij dat ik dit project voor hun heb kunnen doen want hierdoor heb ik echt het gevoel gekregen dat ik met mijn afstuderen waarde voor iemand heb gecreëerd.

Hiernaast wil ik mijn lieve vriendin Xandra bedanken, dat ze het met me uitgehouden heeft omdat ik de afgelopen tijd erg druk ben geweest met dit project.

Tenslotte mijn ouders, bedankt voor de jarenlange steun.

Inhoudsopgave

Hoofdstuk 1

Introductie

Tijdens mijn afstudeerstage bij Fullmoon Interactive Solutions in Amsterdam merkte ik op dat er af en toe situaties ontstonden waarbij het samenwerken niet helemaal goed verliep. Dit kwam voornamelijk doordat mensen aan hetzelfde project bezig waren en hierbij elkaars werk per ongeluk hadden overschreven. Ook was er geen goed beeld wie nou precies waar mee bezig was en wat ze gedaan hadden. Het overzicht ontbrak. Omdat Fullmoon recent meer mensen had aangetrokken en op dat moment bestond uit acht werknemers, moest er eigenlijk goed gekeken worden naar hun manier van werken om te zien of deze nog toereikend was.

Omdat ik uit eigen interesse al veel ervaring had opgedaan met verschillende Source Control Management (SCM) systemen en veel bezig was om voor mijzelf een optimale workflow te creëren stelde ik voor om Fullmoon met hun situatie te helpen. Omdat ik toen nog midden in mijn stagewerkzaamheden zat hebben we besloten om hier een afstudeerproject van te maken, zodat ik mij er volledig op kon focussen. Doordat ik vanuit mijn opleiding Interactieve Media nooit in aanraking ben gekomen met zaken als SCM en professionele workflows leek het me nuttig om mijn ervaringen op te schrijven, waarvan dit document het resultaat is.

De manieren waarop ik samen met Fullmoon hun situatie heb kunnen verbeteren zijn ook voor andere bedrijven en Interactieve Media studenten toepasbaar. Daarom hoop ik dat ze er met dit document hun voordeel mee kunnen doen.

Hoofdstuk 2

Eisen professionaliteit

2.1 Introductie

Voordat er adviezen gegeven kunnen worden voor een goede werksituatie moet eerst in kaart gebracht worden wat de eisen waren bij Fullmoon om goed professioneel te kunnen werken. De situatie hier is dus gericht op het midden- en kleinbedrijf (MKB), maar is in mindere mate ook zeker toepasbaar voor kleine groepen maar ook voor mensen die zelfstandig aan een project werken.

Naast deze eisen volgt nog een lijst van bijkomende zaken die niet direct vereist zijn maar wel erg handig kunnen blijken voor een professionele manier van werken.

2.2 Eisen

1. Medewerkers moeten gelijktijdig kunnen samenwerken aan hetzelfde project zonder elkaar hierbij in de weg te zitten.

2. Er moet een overzicht zijn van wat er gebeurt: wie heeft wat aangepast en wanneer, waar is iedereen mee bezig?

3. Er moet een goede ontwikkelsituatie zijn – gescheiden in verschillende omgevingen – gebaseerd op een OTAP [1] straat.

4. Van alle werkzaamheden moeten goede backups gemaakt kunnen worden. Dit van zowel de productie- als ontwikkelomgevingen. Er moet geen versie van een bestand verloren kunnen gaan en men moet altijd terug kunnen keren naar een vorige versie van een bestand.

[1] Ontwikkel, Test, Acceptatie, Productie

5. Klanten moeten in het ontwikkelproces betrokken kunnen worden, zodat nieuwe projecten en aanpassingen beoordeeld kunnen worden in een omgeving die los staat van de productieomgeving.

2.3 Handige zaken

- Een ticketsysteem dat intern en extern bereikbaar is, zodat werknemers (en klanten) op een centrale plaats kunnen noteren wat er mis is aan een bepaald project en wat er nog moet gebeuren.
- Een eenvoudige manier om projecten door de verschillende omgevingen heen te loodsen zodat bestanden niet allemaal met de hand geupload hoeven te worden.

Hoofdstuk 3

Advies professionaliteit

3.1 Introductie

Om aan de opgestelde eisen van professionaliteit te kunnen voldoen volgen er in dit hoofdstuk een aantal adviezen. Deze worden kort benoemd waarna er in hoofdstuk 4 dieper op ingegaan wordt.

3.2 Adviezen

1. Gebruik maken van een SCM systeem. Dit faciliteert gelijktijdig het beter kunnen samenwerken en zorgt er bovendien voor dat er een historie aanwezig is van veranderingen aan bestanden. Dit is ook een onmisbaar onderdeel om een goed overzicht te krijgen van wat iedereen doet.

2. Medewerkers laten ontwikkelen op hun eigen workstations in plaats van op een centrale server. In samenwerking met het SCM systeem zal iedereen onafhankelijk van elkaar kunnen werken en toch wijzigingen van anderen kunnen binnenhalen.

3. Een ontwikkelstraat gebruiken zodat een project meerdere omgevingen doorloopt. Naast de voordelen die dit biedt voor samenwerking en kwaliteitscontrole kan ook de klant in het ontwikkelproces betrokken worden door bijvoorbeeld wijzigingen uit te testen alvorens deze online komen.

4. Er moet een goede mogelijkheid zijn om backups te maken zodat er geen informatie verloren kan gaan. Dit geldt voor zowel normale bestanden als informatie uit een database.

5. Er moet een centraal support/ticket systeem komen waar zowel medewerkers als klanten toegang toe hebben. Hier kunnen bijvoorbeeld bugs en

taken op aangemeld worden, die vervolgens toegewezen worden aan medewerkers. Zo heeft men een goed overzicht van taken en wordt er een transparant proces voor de klant geboden.

6. Het updaten ("deployen") van sites moet eenvoudig verlopen om er voor te zorgen dat het proces medewerkers niet in de weg zit en zodat fouten die kunnen ontstaan tijdens het handmatig updaten vermeden kunnen worden.

Hoofdstuk 4

Uitgewerkte adviezen

4.1 Introductie

In dit gedeelte van het document wordt dieper op de eerder opgestelde adviezen ingegaan. In sommige gevallen worden er een aantal opties aangedragen waarna de persoonlijke voorkeur van de auteur wordt uitgesproken.

N.B. Om de adviezen toegankelijk te houden voor de meeste bedrijven en studenten zijn de adviezen en keuzes gebaseerd op open source software.

4.2 SCM systeem

4.2.1 Voordelen

Gebruik maken van een SCM systeem biedt ontzettend veel voordelen. Het opslaan van verschillende versies van bestanden tijdens het ontwikkelproces is van onmisbaar belang. Het biedt veiligheid want men kan altijd terugkeren naar een vorige versie, wanneer er bijvoorbeeld een fout in de code is geslopen. Het zorgt er ook voor dat men goed kan samenwerken omdat elke werknemer individueel kan werken aan zijn of haar eigen versie van een project. Hierdoor kan het ook een overzicht bieden van wie wat doet, men kan precies zien wat is aangepast in een bepaald bestand en door wie.

Naast deze – op zich voor de hand liggende voordelen – biedt het ook stabiliteit. Doordat met een SCM systeem de mogelijkheid hebt om te *branchen* is gelijktijdig werken met meerdere versies van een project mogelijk. Bijvoorbeeld *stable* voor de stabiele versie en een *development* versie waarin gewerkt wordt. Doordat wijzigingen in een speciale ontwikkel branch worden uitgevoerd, blijft je stabiele branch onaangeraakt en daardoor dus altijd stabiel. In een ideale situatie worden alle wij-

zigingen uitvoerig getest voordat ze in de stabiele versie worden geïntegreerd. Omdat je dus kan werken in een gescheiden branch is het veilig om te experimenteren, hetgeen in mijn ogen een groot voordeel is.

4.2.2 Tools

Tegenwoordig zijn er tientallen SCM systemen om uit te kiezen. Een greep uit het aanbod: *Subversion, Git, Mercurial* en *Bazaar*. Eigenlijk bieden ze allemaal ongeveer dezelfde functionaliteit, zodat de keuze om hier één van uit te selecteren van andere factoren zal afhangen.

Van dit rijtje is Subversion eigenlijk van de oude school. Het is een systeem dat gebaseerd is op een centrale *repository*, waarin alle wijzigingen worden bijgehouden. De andere behoren tot de zogenaamde DVCS familie, oftewel Distributed Concurrent Version System. Dit systeem heeft als voordeel dat er geen centrale server nodig is.

Bij een DVCS draait iedereen immers een lokale repository op zijn eigen computer. Bij Subversion moet wel gebruik gemaakt worden van één centrale server, waarmee iedereen verbinding moet maken om te kunnen werken. Omdat je in het geval van een DVCS niet afhankelijk bent van een server kun je dus offline en onderweg werken. Hiernaast is het een stuk gemakkelijker en sneller om branches te maken omdat alles lokaal gebeurt, in plaats van over het netwerk. Bij Subversion is dat een tijdrovend bezigheid, terwijl bij een systeem als Git het maken van dezelfde branch slechts enkele seconden zal duren in plaats van minuten. Tijd waarin je niet met het project kunt werken. Doordat het een snel en eenvoudig proces is, wordt je aangemoedigd tot experimenteren (je kunt immers snel even een branch aanmaken om iets uit te proberen) en word je niet uit je workflow gehaald, wat met Subversion wel het geval zou zijn.

In de meeste gevallen richt je een DVCS ook zo in dat iedereen wijzigingen verstuurt (*pushed*) naar een centrale server, hetgeen zorgt voor een goed overzicht. Doordat iedereen zelf kiest wat er naar deze server wordt gepushed houd je een nette repository. Deze wordt niet vervuild door wijzigingen die bijvoorbeeld verkeerd waren of waar iemand nog niet mee klaar is.

Een DVCS heeft door bovenstaande mogelijkheden al mijn voorkeur, maar Subversion heeft ook iets wat in haar voordeel speelt. Doordat het van de genoemde tools de oudste is, zijn er erg veel verschillende applicaties en pakketten voor beschikbaar, waaronder veel grafische programma's. In tegenstelling tot de andere waarbij het tot nu toe meer een commandline aangelegenheid blijft.

Wanneer je alleen naar de gedistribueerde pakketten bekijkt is er een aantal voordelen waarin Git zich onderscheidt van de rest. Deze zijn als volgt.

- Goedkope local branching. Git geeft de mogelijkheid zoveel lokale branches aan te maken als men wil, waarbij het creëren, mergen en deleten hiervan

slechts enkele seconde duurt. De andere systemen raden aan dat de beste branch een kloon is van de repository in een andere directory en bieden niet deze handigheid.

- Git is ontzettend snel[1]. Omdat het ontworpen is voor het werken aan de Linux kernel, waren het gebruik maken van grote repositories en snelheid uitgangspunten tijdens de ontwikkeling, en dat is goed te merken.

- Git leent zichzelf voor elke workflow. Het heeft niet één bepaald workflow model meegekregen, dus is het naar eigen inzicht te gebruiken. Bijvoorbeeld in de vorm van een Subversion-style systeem waarbij iedereen zijn wijzigingen naar een centrale server pushed. Maar ook de zogenaamde *Integration Manager* workflow is mogelijk, waarbij er één persoon verantwoordelijk is om alle wijzigingen in een project door te voeren.

Advies: Als men niet wordt afgeschrikt door de commandline wordt aangeraden gebruik te maken van Git. Maakt men liever gebruik van een goede grafische tool dan kan men het beste voor Subversion kiezen.

4.3 Ontwikkelstraat

De meeste bedrijven zullen hopelijk al gebruik maken van twee ontwikkelomgevingen, een waarin men zelf werkt en de live website, die bezoekers te zien krijgen. In een professionele situatie zal dit uitgebreid moeten worden naar het liefst vier verschillende omgevingen, ontwikkeling, test, acceptatie en productie, oftewel de al eerder genoemde OTAP-straat.

4.3.1 Ontwikkeling

Als een bedrijf gebruik maakt van één centrale server in de ontwikkelomgeving is het van belang dat deze situatie veranderd naar één waarin de werknemers ontwikkelen op hun eigen computers. Dit om gebruik te kunnen maken van een SCM systeem. Op deze manier kan iedereen individueel werken, zonder dat ze last van elkaar ondervinden doordat ze aan hetzelfde project bezig zijn.

In de ontwikkelfase, wanneer mensen bijvoorbeeld schermschetsen en HTML pagina's gaan maken zal dit nog niet veel veranderen aan de workflow. Maar op het moment dat er een website geprogrammeerd moet worden zal het in de meeste gevallen nodig zijn om lokaal ook een webserver te draaien.

Omdat er veel diversiteit is in programmeertalen en operating systems, is het onbegonnen werk om hier alle manieren te vermelden om dit te bereiken, maar er is wel iets waar op gelet moet worden. Zorg er voor dat alle versies van software

die op de servers en workstations draaien overeen komen. Het gaat hierbij dan om zaken als de web- en databaseserver en bijvoorbeeld PHP. Stel dat een bug in een website opgelost moet worden, als de workstations een afwijkende versie draaien van bijvoorbeeld de webserver – waarin deze bug niet voor kan komen – wordt het moeilijk om deze op te lossen.

4.3.2 Test

De testomgeving is een interne webserver die een bepaalde versie van het project draait, zodat iedereen deze daar kan testen. Dit vormt ook een centrale plek om projecten te kunnen bekijken zodat je niet alles hoeft binnen te halen, wat vooral handig is voor mensen die niet direct met het project te maken hebben.

Voor kleine bedrijven kan een specifieke testomgeving een beetje overkill vormen, maar voor grotere bedrijven waar bepaalde mensen verantwoordelijk zijn voor het testen en QA (Quality Assurance) kan het erg handig zijn.

4.3.3 Acceptatie

De acceptatieomgeving is een afgeschermde webserver die wel toegankelijk is voor de klant. Deze kan hierop een bepaalde versie van het project bekijken en hier feedback op geven. Wanneer iemand bijvoorbeeld de taak heeft gekregen om nieuwe functionaliteit aan een website toe te voegen is het van belang dat de gebruiker deze pas ziet wanneer het helemaal klaar en goedgekeurd is. Het is tamelijk onprofessioneel als bezoekers een niet werkende site te zien krijgen, omdat er live aan gewerkt wordt.

4.3.4 Productie

Ten slotte heb je de productieomgeving: de live website, dit spreekt voor zich.

Advies: Maak gebruik van meerdere ontwikkelomgevingen, zodat de klant in een project betrokken kan worden en bezoekers van de website geen hinder ondervinden van de werkzaamheden. Let er hierbij op dat belangrijke softwareversies overeen komen in de verschillende omgevingen.

4.4 Backup

Indien er gebruik gemaakt wordt van SCM is de backup van broncode in principe al vrij goed geregeld. Iedere versie van een bestand wordt opgeslagen en men kan dus altijd terug naar een vorige versie. Het regelmatig maken van backups wordt gelukkig door de meeste mensen al op waarde geschat, maar vaak genoeg zie je

dat deze backups bewaard worden op hetzelfde systeem. Wanneer deze onverhoopt ten onder gaat ben je alsnog alles kwijt. Zo heb je een *single point of failure* gecreëerd.

Om een echt goede backupsituatie te hebben moet deze ten minste op een andere machine of bijvoorbeeld een externe harde schijf worden bewaard. Het verdient de voorkeur om de backup ook nog op een fysiek andere locatie te hebben, want in geval van brand is niets veilig.

Tegenwoordig is hosting in de "cloud" in opkomst, waarin Amazon voorop loopt met haar Simple Storage Service (S3). Een account bij S3 is een betrekkelijk voordelige manier om veel data op te slaan. Er zijn inmiddels ook al veel tools voor beschikbaar om het maken van backups gemakkelijk te laten verlopen. Wanneer men er voor kan zorgen dat backups automatisch op vaste tijdstippen worden gemaakt geeft het een gevoel van veiligheid en is het tijdsbesparend. Een groot voordeel van S3 is dat alles daar ook nog eens gemirrored wordt. Alle bestanden op hun systeem worden namelijk op diverse locaties in de wereld opgeslagen. Op deze manier heb je enorm veel zekerheid over je backups.

Als men er voor kiest om gebruik te maken van S3 zijn er veel tools waar je uit kan kiezen.[2] De voorkeur van de auteur ligt bij s3sync[3], wat het eenvoudig maakt om uploads te automatiseren door middel van scripts.

4.4.1 Database backups

Bij de meeste hostingproviders kan gebruik gemaakt worden van hun backupservice om databases te archiveren. Maar op het moment dat deze nodig zijn moet er vaak contact opgenomen worden met de support-afdeling. Bij voorkeur verzorgt men deze backups dus ook zelf, om niet afhankelijk te zijn van zulke externe factoren.

Er zijn verschillende manieren om backups te maken van een MySQL database. Let wel, bijna al deze manieren moeten MYSQL stoppen (of zetten hier een lock op) om ervoor te zorgen dat de inhoud van de databases niet verandert tijdens de backup. Backups maken doe je dus bij voorkeur op tijden dat de site nauwelijks bezocht wordt. Een andere manier is om een *master-slave* systeem in te voeren, waarbij gebruik gemaakt wordt van twee verschillende databases. Hierbij kan de slave database realtime of op gezette tijden gesynchroniseerd worden met de master database. De backup kan dan vervolgens worden uitgevoerd op de slave, die zonder problemen gestopt kan worden. Waardoor de master database altijd online blijft en de bezoekers geen hinder ondervinden van het backupproces.

MYSQL levert mysqldump mee, waardoor het triviaal maakt om backups te maken. In de bijlagen is een eenvoudig script bijgevoegd dat gebruik maakt van mysqldump en s3sync. Hiermee kan elke avond een backup gemaakt worden die naar Amazon S3 wordt geupload.

Advies: Zorg voor regelmatige backups en bewaar deze op meerdere plaatsen zodat er altijd zekerheid is over de veiligheid van data.

4.5 Ticketsysteem

Wanneer een bedrijf begint te groeien en met veel klanten te maken krijgt, kan het leveren van support en het overzien van nieuwe taken werknemers overmeesteren. Bugs en verzoeken (issues) doorkrijgen via de mail kan een tijd lang goed blijven gaan, maar wanneer er bijvoorbeeld iemand ziek is en deze persoon de enige ontvanger was van deze mails heb je dus net zoals bij punt 4.5 een *single point of failure*. Door een centraal overzicht te creëren sluit je situaties als deze uit en is de informatie voor iedereen beschikbaar.

Men kan er voor kiezen om dit ticketsysteem alleen intern te gebruiken, maar de transparantie van een bedrijf kan er mee verbeterd worden wanneer de klanten ook toegang hebben. Zij kunnen dan in één oogopslag zien hoe het met hun tickets gaat en hier weer op reageren. Zo blijven bijvoorbeeld supportaanvragen niet verstopt in de e-mail van één werknemer maar is alles algemeen beschikbaar.

4.5.1 Voordelen

Naast de al reeds genoemde voordelen van een centraal overzicht van supportaanvragen en transparantie, zijn er nog enkele voordelen te noemen.

Voordelen voor medewerkers:

- Goed overzicht van taken waar aan gewerkt moet worden en waar aan gewerkt is.
- Prestatiebevorderend omdat het zichtbaar wordt gemaakt wat iemand heeft gedaan om iets op te lossen. Een ticket sluiten geeft erg veel voldoening.

Voordelen voor het bedrijf:

- Werknemers kunnen goed en concreet aangestuurd worden door ze issues toe te wijzen.
- Bij de tickets kan goed bijgehouden worden hoeveel uur ergens aan is besteed en wat er is gebeurd. Er is dus een centrale plek waar wordt opgeslagen wie en hoe lang aan welk project heeft gewerkt, zodat de gemaakte uren professioneel verantwoord kunnen worden bij de opdrachtgever.
- De prestaties van werknemers wordt in zekere mate meetbaar.

4.5.2 Keuze

Er is werkelijk een overvloed aan aanbod van issue tracking systemen[4] dus er is altijd wel een pakket te vinden dat aan alle specifieke eisen van een bedrijf voldoet. Wel wordt sterk aangeraden om bij de de keuze van een systeem te kijken of dit ook integratie biedt met een SCM systeem. Zo heb je één plek om een overzicht te krijgen van wat er allemaal gebeurt binnen het bedrijf, wijzigingen in de code, openen en sluiten van issues, etc. Ook kan men zo een koppeling leggen door het sluiten van tickets vanuit commitmessages mogelijk te maken. Zodat men snel kan terugzien hoe een bepaalde issue precies is opgelost.

Bij Fullmoon hebben we uiteindelijk voor Redmine[5] gekozen. Naast issue-tracking biedt het een goed overzicht van activiteit per project en algemeen, solide integratie met veel SCM systemen en een geïntegreerd wiki systeem.

Advies: Maak gebruik van een ticketsysteem om support voor klanten en taken voor medewerkers centraal op te slaan.

4.6 Automatisch deployen

In navolging van onderdeel 4.4, het werken met een ontwikkelstraat, is het van belang dat het deployen van code naar verschillende omgevingen soepel en gestroomlijnd gaat. Wanneer men telkens alles handmatig moet uploaden wordt de drempel namelijk te hoog, en gaat dit ten koste van de workflow.

Er zijn een aantal softwareoplossingen om het deployen te stroomlijnen. Veel hiervan zijn zogeheten build tools, zoals Maven[6] en Phing[7]. Dit zijn erg uitgebreide pakketten die het eenvoudig maken om code te bouwen (*compilen*). Omdat dit niet echt aan de orde zal zijn voor de meeste webdevelopment bedrijven en omdat het vrij omslachtig is zal de verdere uitwerking van dit onderdeel gebaseerd zijn op een tool die beter geschikt is, namelijk Capistrano[8].

Capistrano is een tool geboren in de Ruby on Rails community en is bij uitstek geschikt voor het deployen van nieuwe code. Het koppelt erg goed met een SCM systeem en neemt veel werk uit handen. Hiernaast biedt het de gebruiker de mogelijkheid om een *rollback* uit te voeren, wanneer de gedeployde code niet goed blijkt te zijn. Een ander voordeel is het feit dat de eindgebruiker de nieuwe website pas te zien krijgt wanneer deze in zijn geheel online is. Op deze wijze kunnen er geen bugs ontstaan doordat een nieuwe wijziging pas voor de helft is geupload.

De tool wordt voornamelijk gebruikt om nieuwe versies van Ruby on Rails applicaties te deployen, maar het is na een kleine aanpassing ook uitermate geschikt voor bijvoorbeeld PHP applicaties. Dit doe je door in de root van een project een map genaamd config aan te maken, waarin zich een `deploy.rb` bestand bevindt.

Dit is het "recept" dat Capistrano gaat gebruiken om zijn werk te kunnen doen. Hierin staan bijvoorbeeld de locaties van de web- en SCM server, naar welke map alles geupload moet worden, etc.

Met behulp van de Capistrano Multistage plugin[9] kan men verschillende stages (omgevingen) aangeven. Dus bijvoorbeeld acceptatie of productie. Als alles goed is ingesteld kan met één commando code naar de gewenste omgeving worden gedeployed. Om code naar de acceptatie omgeving te uploaden kan bijvoorbeeld dit ingevoerd worden:

```
cap acceptatie deploy
```

In de bijlagen is het recept te vinden dat nu gebruikt wordt bij Fullmoon om hun PHP applicaties succesvol naar verschillende omgevingen te kunnen deployen.

Advies: Er wordt aangeraden om gebruik te maken van Capistrano om het deployen van code naar verschillende omgevingen te vergemakkelijken.

Hoofdstuk 5

Conclusie

Met dit document heb ik de succesvolle workflow verandering bij Fullmoon Interactive Solutions om willen zetten in een document dat voor meerdere mensen bruikbaar is. De werknemers bij Fullmoon kunnen inmiddels zonder problemen samenwerken aan hetzelfde project en er is een centrale plek waar ze een goed overzicht kunnen krijgen van werkzaamheden en taken.

Ik heb mijn best gedaan om een aantal goede adviezen op te stellen en ben van mening dat deze echt iets kunnen toevoegen aan de verdere professionalisering van bedrijven en studenten. Met goede mogelijkheden tot samenwerking door verschillende ontwikkelomgevingen en SCM, een overzicht van werkzaamheden en taken, de optie om klanten in het ontwikkelproces te betrekken en de zekerheid van goede backups wordt een goede bedrijfsvoering mogelijk.

Als er nog niet met een SCM systeem gewerkt wordt, zal dat de grootste en ook meest merkbare verandering zijn. Uit eigen ervaring kan ik zeggen dat het invoeren hiervan echt een nieuwe impuls geeft aan het plezier van werken. Het geeft je een goed gevoel om te kunnen zien wat je precies gedaan hebt en het geeft je zekerheid. Sinds ik gebruik maak van source control werk ik zonder zorgen aan aan websites, met in mijn achterhoofd dat ik geen fouten kan maken aangezien ik altijd een stabiele versie heb. Bij Fullmoon heerst dit gevoel ook.

Ook al draait een bedrijf op dit moment goed, er moet toch naar de toekomst gekeken worden. Stel dat er in het komende jaar drie mensen worden aangenomen, is de huidige manier van werken dan nog wel toereikend? Fullmoon heeft er goed aan gedaan om de situatie nu aan te passen, en is klaar voor de toekomst en om verder uit te groeien.

Studenten heb ik hopelijk met dit document nieuwe ideeën en inspiratie gegeven om ook naar hun eigen manier van werken te kijken om te zien of en hoe ze deze kunnen verbeteren.

Bibliografie

[1] S. Chacon, http://whygitisbetterthanx.com.

[2] J. Zawodny, http://jeremy.zawodny.com/blog/archives/007641.html.

[3] http://s3sync.net/wiki.

[4] Wikipedia, http://en.wikipedia.org/wiki/comparison_of_issue_tracking_systems.

[5] J.-P. Lang, http://www.redmine.org/.

[6] A. Foundation, http://maven.apache.org/.

[7] H. Lellelid, http://phing.info/trac/.

[8] J. Buck, http://www.capify.org/.

[9] J. Buck, http://weblog.jamisbuck.org/2007/7/23/capistrano-multistage.

[10] HackerNews, http://news.ycombinator.com/item?id=374390.

[11] T. Swicegood, *Pragmatic Version Control Using Git* (Pragmatic Programmers, 2008).

[12] M. Mason, *Pragmatic Version Control using Subversion*, 2nd edition ed. (Pragmatic Programmers, 2006).

Hoofdstuk 6

Bijlagen

Op http://www.ikbenbitterzoet.com/weblog/afstudeer/ is een zipbestand terug te vinden welke alle bestanden die hier vermeld worden bevat.

6.1 Capistrano

Wat volgt is het recept wat bij Fullmoon gebruikt wordt om php applicaties naar de acceptatie en productie omgevingen te deployen.

config/deploy.rb

```
require 'capistrano/ext/multistage'

set :application, "APPLICATIE"
set :user, "USER"
set :repository,  "REPOSITORY"
set :deploy_to, "/usr/local/wwwroot/#{application}"
set :deploy_via, "remote_cache"

ssh_options[:forward_agent] = true
ssh_options[:username] = "SSHUSER"
set :use_sudo, false

set :default_stage, "production"
set :stages, %w(production staging)

namespace :deploy do

  # launch the following task after updating the code
```

```
  after "deploy:update_code", "deploy:link_settings"

  desc "Link the settings file to the release directory"
  task :link_settings do
    run "ln -nfs #{deploy_to}/shared/xsettings/settings.php \
    #{release_path}/xsettings/settings.php"
  end

  [:start, :stop].each do |t|
    desc "#{t} task is not necessary with PHP"
    task t, :roles => :app do ; end
  end
end
```

6.2 Backup script

Dit is een voorbeeld van een backup script wat elke avond gedraaid kan worden om een backup van alle databases te genereren waarna deze gelijk naar S3 verstuurd worden.

```
#!/bin/bash

USERNAME=""
PASSWORD=""

S3_BUCKET=""

BACKUP_DIR="$(date +%F)"

mysqldump -u $USERNAME -p$PASSWORD --all-databases | gzip > $BACKUP_DIR.sql.gz

/Users/markmulder/backup/s3sync/s3sync.rb --recursive $BACKUP_DIR $S3_BUCKET
```