GIT WORKFLOW LECTORAAT

ONDERZOEKSRAPPORT

Marco Verbist

GIT WORKFLOW LECTORAAT

ONDERZOEKSRAPPORT

Marco Verbist

Plaats van uitgave:	Vlissingen
Datum:	16-05-2016
Student:	Marco Verbist
Studentnummer:	00062951
Cursus:	VCC
Docent:	Wouter Everse
Versie:	1.2

Inhoudsopgave

Revisiegeschiedenis

Onderstaande tabel geeft een overzicht van de revisiegeschiedenis waarmee dit document tot stand is gekomen. Per revisie is aangegeven wat de wijzigingen zijn.

Rev.	Datum	Auteur(s)	Wijziging(en)
1.0	28-01-2016	M.V.	• Initiële documentstructuur opgezet • H1 Inleiding ingevuld • H2 Achtergrond ingevuld • H3 Resultaten: 3.1 ingevuld
1.0.1	02-02-2016	M.V.	• Voorpagina toegevoegd • H3 Resultaten: 3.2 ingevuld • Inhoudsopgave bijgewerkt
1.1	22-03-2016	M.V.	• Documentopmaak herzien • H3 Resultaten aangevuld • Advies uitgebracht • Discussie toegevoegd • Conclusie toegevoegd • Vorm aangepast (Portfolio)
1.2	09-05-2016	M.V.	• Bronnenhoofdstuk toegevoegd (bibliografie) • Bronnen toegevoegd aan resultaten • Workflows uitgebreider gemaakt: uitwerking beschreven en illustraties toegevoegd • Vergelijking toegevoegd voor workflows • Voor- en nadelen beschreven voor workflows • Zelfbeoordelingen volledig herschreven • Inhoudsopgave bijgewerkt

1. Inleiding

In het vierde jaar van de opleiding HBO-ICT aan de University of Applied Sciences (voorheen Hogeschool Zeeland) is het gebruikelijk dat studenten een klein aantal EC's (studiepunten) verwerven middels het VCC-programma, wat staat voor Vrije Compositie Cursus

Na de beschikbare opties bekeken te hebben heb ik de voorkeur gegeven aan een individueel project. Docent Wouter Everse had vanuit het Lectoraat een passend project, gelieerd aan het gebruik van GIT als versiebeheerprogramma binnen de ontwikkeling van een *knowledge database*.

Middels dit portfolio zal ik duidelijk maken wat er van mij werd verwacht, wat mijn leerdoelen waren, op welke wijze ik deze heb bereikt en uiteraard de bijhorende bewijslast (inhoud) van het geheel.

2. Achtergrond

2.1 Aanleiding

Binnen het Lectoraat van de University of Applied Sciences is men bezig met het ontwikkelen van een *knowledge database* in de vorm van een Mediawiki systeem.

Concreet werken er meerdere ontwikkelaars afwisselend aan dit systeem, daarbij wordt er gebruik gemaakt van GIT als versiebeheersysteem.

Binnen het ontwikkelteam leeft de gedachte dat men wellicht niet alles uit GIT haalt c.q. GIT wellicht niet op de juiste manier gebruikt voor de context.

2.2 Doelstelling & Leerdoelen

Middels een aantal vooraf opgestelde leerdoelen ga ik proberen het Lectoraat een zo goed mogelijk advies te geven wat betreft het gebruik van GIT binnen hun Workflow. Hieronder de vier leerdoelen welke ik middels dit project wil behalen:

1. De basisprincipes van het versiebeheerprogramma GIT onderzoeken & begrijpen door te experimenteren zodat ik spoedig in staat ben GIT te implementeren en gebruiken in een nieuw project
2. Kennis opdoen van de diverse manieren waarop GIT kan worden gebruikt – middels zogenaamde GIT Workflows – door deze te implementeren in een testproject.
3. Onderzoek doen naar de huidige werkwijze binnen het Lectoraat door interviews af te nemen met de betrokken personen, daarnaast de huidige rol van GIT hierin herkennen en benoemen
4. In kaart brengen wat een gewenste werkwijze is en welke GIT Workflow hier het best bij past, of indien nodig zelf een aangepaste GIT Workflow definiëren

2.3 Methode

Om aan het eind van het onderzoek een goed advies te kunnen geven heb ik een hoofdvraag opgesteld.

Op welke wijze kan de werkwijze (wat betreft gezamenlijk ontwikkelen) van het ontwikkelteam binnen het Lectoraat worden geoptimaliseerd, kijkend naar specifiek GIT als versiebeheersysteem?

Daarnaast zal ik de in Hoofdstuk 2.2 opgestelde leerdoelen gebruiken als deelvragen om zo de hoofdvraag te beantwoorden.

3. Resultaten

3.1 Basisprincipes GIT

Om erachter te komen wat de basisprincipes van GIT zijn ben ik op zoek gegaan naar een down-to-earth introductie van GIT.

De basis van GIT

GIT is een softwareprogramma dat de principes van versiebeheer praktiseert. Concreet houdt dit in dat wanneer er in een project (goed) gebruik wordt gemaakt van GIT alle wijzigingen van alle teamleden in GIT worden bijgehouden. Groot voordeel hiervan is dat er een duidelijke structuur is te onderscheiden en dat men eenvoudig kan terugvallen op een vorige versie van een specifiek bestand, map of zelfs een heel project.

Op het moment dat twee ontwikkelaars wijzigingen aanbrengen in hetzelfde bestand herkent GIT dit en is er mogelijkheid tot *merging*. De wijzigingen worden dan naast elkaar tegen het licht gehouden waarna er een beslissing kan worden genomen over hoe deze wijzingen zullen worden samengevoegd.

GIT Repositories

GIT slaat alle bestanden & mappen waarvoor het versiebeheer uitvoert op in een zogenaamde *GIT Repository*, ook wel afgekort als een *GIT Repo*.

Een GIT Repo kan worden verkregen door zelf een Repository te initiëren, of door een bestaande Repo te *clonen*. Op het filesystem is een *GIT Repo* herkenbaar aan de (standaard) verborgen *.git* map.

Het initiëren van een Repo

Het initiëren van een nieuwe GIT Repo is een vrij eenvoudig proces. De gebruiker navigeert naar de gewenste directory, waarna het commando git init wordt uitgevoerd. GIT maakt nu een (standaard) verborgen map aan: .git.

Het clonen van een bestaande Repo

Een bestaand Repository kan eenvoudig worden overgenomen wanneer de locatie bekend is.

```
git clone git://github.com/gebruiker/repository.git
```

of

```
git clone git://github.com/gebruiker/repository.git customDirectory
```

Er wordt nu een directory repository aangemaakt, waarin naast de inhoud ook een .git map word aangemaakt. Door gebruik te maken van het tweede commando heeft de aan te maken directory customDirectory in plaats van repository.

Het toevoegen van bestanden

Zodra de Repository is aangemaakt kan de gebruiker (wanneer nodig) bestanden toevoegen. Dit is mogelijk middels de volgende commando's.

```
git add README test.html
```

```
git add *
```

Middels het eerste commando worden enkel de bestanden README en test.html aan de repo toegevoegd. Middels het tweede commando worden alle bestanden in de huidige directory toegevoegd.

Het bijhouden en *opslaan* van wijzigingen

Wanneer een GIT Repo eenmaal lokaal actief is en wijzigingen bevat, wilt de gebruiker deze op een moment inzien en definitief opslaan – committen, dit kan middels onderstaande commando's.

Het commando git status toont een overzicht van alle *openstaande* bestanden, concreet omvat dit:

- De huidige branch
- Bestanden welke gereed zijn om te worden gecommit
- Bestanden welke nog niet zijn *gestaged* – niet zijn opgenomen in de GIT Repo

Het commando git commit staat de gebruiker toe zijn wijzigingen definitief te maken door ze op te slaan in de GIT Repo. Wanneer het commando wordt uitgevoerd zoals hierboven opent de standaard editor, zodat de gebruiker commentaar voor de commit kan meegeven. Wanneer de parameter *m* wordt meegegeven kan er direct commentaar worden meegegeven:

```
git commit –m "Bugfix for bug #1432".
```

Bestanden verwijderen

Het kan voorkomen dat het wenselijk is om een bepaald bestand te verwijderen uit de GIT Repo. Dit kan middels het commando git rm myFile.txt.

Het aangegeven bestand zal nu worden verwijderd uit GIT, wanneer het git status commando wordt uitgevoerd zal worden aangegeven dat het bestand *deleted* is.

Bestanden hernoemen & verplaatsen

Wanneer men een bestand wilt verplaatsen is dit eenvoudig mogelijk met een specifiek GIT-commando dat veel overeenkomt met het native Linux commando.

```
git mv myFile.txt path/to/other/directory/myFile.txt
```

of

```
git mv myFile.txt newFilename.txt
```

Wanneer nu het commando git status zal worden uitgevoerd zal worden aangegeven dat het bestand *renamed* is, dit is de manier hoe GIT omgaat met het verplaatsen van bestanden.

GIT Historie bekijken

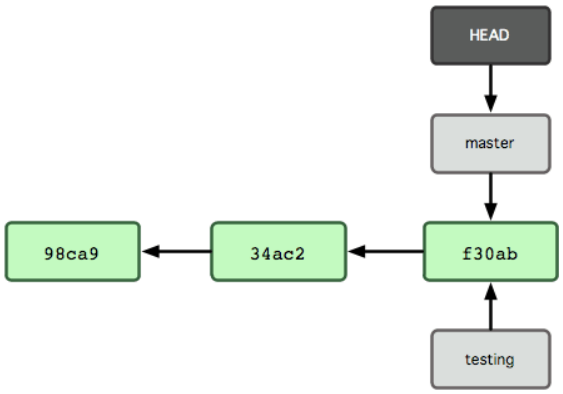
Wanneer een project geruime tijd loopt en vooral wanneer er veel ontwikkelaars aan werken ontstaat er een grote geschiedenis.

Middels het commando git log wordt er een overzicht getoond van alle uitgevoerde acties sinds het begin van de GIT Repo.

Branches

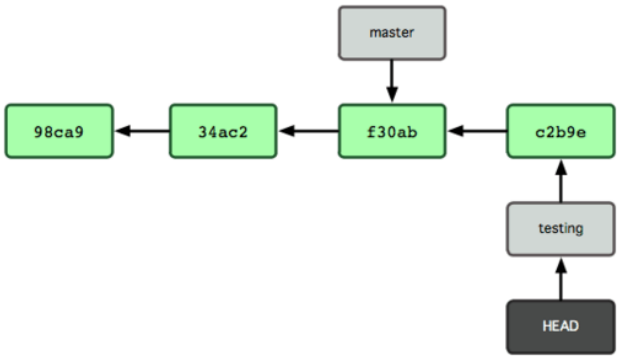
Branches zijn een fundamenteel onderdeel van GIT, een branch is het best te omschrijven als een verwijzing naar een momentopname van een serie bestanden binnen GIT.

De standaard-branch binnen GIT heet **master**, hierin zullen standaard dan ook alle commits 'plaatsvinden'.



Bovenstaande illustratie verduidelijkt het gebruik van branches binnen GIT wellicht. Allereerst zijn er drie commits uitgevoerd, waarnaar wordt verwezen met de namen **98ca9**, **34ac2** en **f30ab**.

Na de laatste commit is er een nieuwe branch **testing** aangemaakt, deze heeft dus net als de **master** branch alle commits met zich mee. De **HEAD**-pointer geeft aan welke branch momenteel in gebruik is. In bovenstaande illustratie is er dus een nieuwe branch aangemaakt, maar deze is niet actief.



De tweede illustratie geeft weer wat er gebeurt wanneer je de testing branch als actief instelt en hierop verder gaat werken. Nieuwe commits zullen worden toegevoegd aan de testing branch, waardoor de master branch intact blijft.

Zoals in de illustratie is gedaan kan een bepaalde test in een aparte branch worden opgeslagen, zonder dat de master branch daardoor wordt aangetast. Wanneer er weer terug gewisseld wordt naar de master branch zou commit **f30ab** weer het meest recent zijn.

Branches mergen

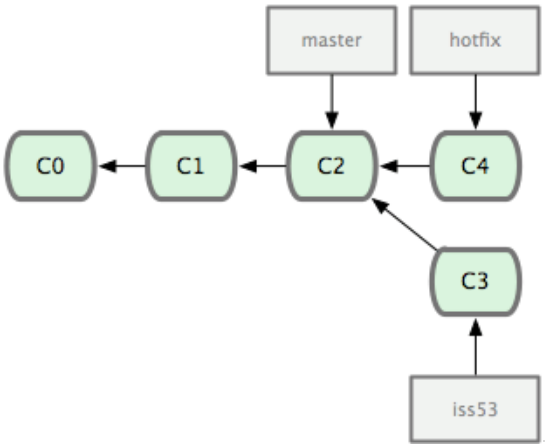
Om de werking van het mergen te verduidelijken zal ik allereerst een realistisch scenario schetsen.

Je werkt aan een project met een GIT Repository. Voor nieuwe functionaliteit maak je een nieuwe branche ***experiment*** *aan, dus een aftakking van de* ***master*** *branch van dat moment. Er komt tussendoor een dingend verzoek voor een bugfix, je switcht terug naar de* ***master*** *branch en maakt een nieuwe branch* ***bugfix*** *aan. In deze branch implementeer je een oplossing voor de bug. Zodra de fix getest is meng je de branches* ***master*** *en* ***bugfix****, je voegt eigenlijk de inhoud van* ***bugfix*** *toe aan* ***master****.*

De afbeelding hiernaast verduidelijkt het geheel. Op dit punt is er een aftakking van **master** gemaakt – **hotfix** – waarin de fix is geïmplementeerd.

Om het geheel te mengen – mergen – dient **master** te worden gemerged met **hotfix**.

Binnen GIT kan dit heel eenvoudig met een aantal commando's, hieronder een verduidelijking.



Commando	Omschrijving
`git branch testing`	Maakt de branch **testing** aan
`git checkout testing`	Stelt de **testing** branch als actief in, dit wisselt de bestanden
`git checkout -b testing`	Maakt de branch **testing** aan, en stelt hem in als actief
`git checkout master` `git merge testing`	Stelt de **master** branch als actief in en mengt hem met de **testing** branch in de **master** branch
`git branch -d testing`	Verwijdert de **testing** branch
`git branch -D testing`	Verwijdert de **testing** branch, ook wanneer er nog niet-gedane commits zijn
`git branch -v`	Toont een overzicht van alle branches, de actieve branch heeft een * (asterix) als voorvoegsel; voeg parameter –v toe om de laatste wijzigingen te zien

In bijlage 5.5 is te zien hoe wijzigingen in hetzelfde bestand vanaf twee branches kunnen worden doorgevoerd, deze worden waar mogelijk samengevoegd. Wanneer dit samenvoegen onverhoopt niet lukt is er sprake van een *merge conflict* en opent een editor waarin de gebruiker zelf kan aangeven welke wijzigingen prioriteit hebben boven andere(n).

Bronnen: (Dudler), (Atlassian, Atlassian GIT Tutorials), (git-scm.com, Git - documentation)

3.2 GIT in de praktijk

Om de basisprincipes van GIT in de praktijk te brengen heb ik een basisproject opgezet en hier vervolgens een GIT Repository aan toegevoegd.

Deze uitwerking is te vinden in Bijlage 5.

Bron: (Dudler)

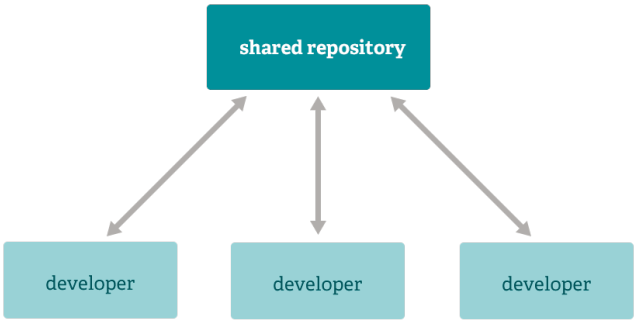
3.3 GIT Workflows

In de praktijk zijn er een aantal methoden waarop GIT gebruikt kan worden in een team: ook wel GIT Workflows genoemd.

Voor dit onderzoek zijn er een aantal workflows bekeken en op een rij gezet.

3.3.1 Centralized Workflow

De Centralized Workflow staat ook wel bekend als de klassieke SVN-methodiek omdat deze vrijwel overeenkomt met hoe SVN werkt. In deze workflow is er één gedeelde repository waar alle ontwikkelaars in werken.

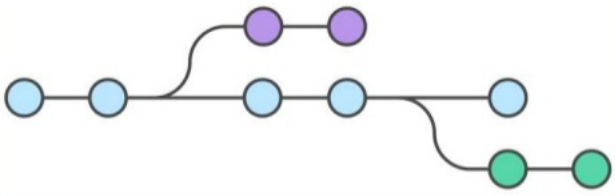


Initieel maakt iedere ontwikkelaar een kopie (clone) van de centrale repository. Vervolgens kan hij of zij lokaal wijzigingen maken. Op het moment dat de wijzigingen gereed zijn voor terugkoppeling naar de centrale repository dient de ontwikkelaar eerst een update uit te voeren zodat eventuele wijzigingen van andere ontwikkelaars zijn opgenomen. Tijdens deze update komen tevens ook eventuele conflicten naar boven. Mocht dit het geval zijn dienen deze te worden opgelost voor de update kan worden uitgevoerd.

Aangezien er in de centrale repository maar één branch (master) aanwezig is, is een update verder niet heel spannend. De lokale repository van de ontwikkelaar wordt simpelweg samengevoegd met de centrale repository waarna de wijzigingen zijn opgenomen. Mocht er bij het updaten een *merge conflict* plaatsvinden dan kan dit op de reguliere wijze worden verholpen.

Zie Bijlage 5.3 voor een uitwerking van deze Workflow.
Bron: (Atlassian, Comparing workflows), (Hodson)

3.3.2 Feature Branch Workflow



De *Feature Branch Workflow* is sterk gebaseerd op de Centralized Workflow, zoals hierboven genoemd. Het concept dat deze workflow uniek maakt is dat men voor iedere *feature* een eigen branch aanmaakt binnen GIT, in de centrale repository. In de praktijk kan zo'n genoemde feature verschillen van een complete nieuwe functionaliteit tot een bugfix.

Door dat er voor iedere feature een eigen branch wordt gehanteerd is het per definitie uitgesloten dat er conflicten ontstaan tussen ontwikkelaars welke ieder aan een eigen stukje functionaliteit werken.

Deze workflow wordt vooral geprijsd omdat ontwikkelaars aan de ene kant ongehinderd en ongelimiteerd code naar de centrale server kunnen pushen zonder anderen tot last te zijn, en het daarnaast op ieder moment mogelijk is om de code van een specifieke feature te mergen met bijvoorbeeld de master branch, of een tussenliggende branch.

Zie Bijlage 5.4 voor een uitwerking van deze Workflow.
Bron: (Atlassian, Comparing workflows), (Driessen, 2010)

3.3.3 Gitflow Workflow

De *Gitflow Workflow* schetst is in de kern gebaseerd op de Feature Branch Workflow. Het grote verschil is dat features zodra ze zijn afgerond niet direct in de master repo worden geplaatst, maar eerst in de tussenliggende development repo.

Deze development repo fungeert als tussen-repo waarin alle feature-branches worden verzameld. Zodra aan een bepaalde voorwaarde wordt voldaan: bijvoorbeeld wanneer een datum is aangebroken (x weken verstreken) of wanneer er x features zijn afgerond wordt er vanuit de development repo een release gemaakt. Deze release is de praktijk een nieuwe branch welke alle features bevatten welke op dat moment in de development repo staan.

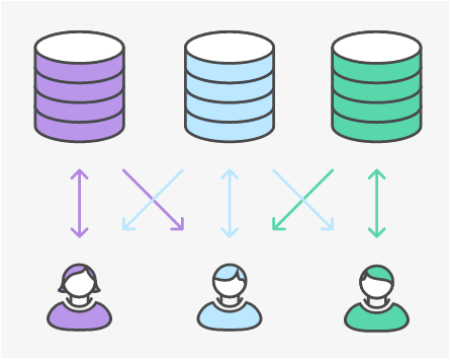
De release wordt van een tag voorzien en vervolgens doorgezet (*merged*) naar de master repo.

Deze workflow maakt het bijhouden van releases vele malen makkelijker. Wanneer er vanwege welke reden dan ook de wens ligt om terug te grijpen op een vorige versie van de complete applicatie is dit heel eenvoudig uitvoerbaar. Terugvallen op de vorige versie van een enkele feature is in deze workflow wat lastiger.

Zie Bijlage 5.5 voor een uitwerking van deze Workflow.
Bron: (Atlassian, Comparing workflows)

3.3.4 Forking Workflow

De *Forking Workflow* is vele malen complexer dan de hiervoor behandelde workflows. Bij de Forking Workflow heeft iedere gebruiker zowel lokaal als op de server een repository. Visueel ziet dat er als volgt uit.



Wat direct opvalt aan de Forking Workflow is dat uiteindelijk slechts één persoon in staat is om de uiteindelijke master-repo bij te werken. Iedere medewerker heeft een clone van zijn repo op de server en is vrij om lokaal aan features te werken. Op het moment dat de individuele wijzigingen zijn afgerond worden ze ge-commit naar de repo op de server.

Dit is het punt waar de beheerder om de hoek komt kijken: de beheerder neemt gedeeltelijk of volledig de wijzigingen van de ontwikkelaars over in de uiteindelijke master-repo. Indien wenselijk kan de beheerder ervoor kiezen om ook in de master-repo gebruik te maken van *branching*, wat betreft structuur kan worden gekozen voor een structuur zoals deze in de voorgaande workflows is behandeld. Zo kan de beheerder er bijvoorbeeld voor kiezen om in de master-repo release-branches aan te houden.

Zie Bijlage 5.6 voor een uitwerking van deze Workflow.
Bron: (Atlassian, Comparing workflows), (Git Workflows)

3.3.5 GIT Submodules

In dit hoofdstuk wordt geen GIT Workflow behandeld, maar een functionaliteit uit GIT welke mogelijk relevant is binnen deze context.

Een GIT Submodule maakt het mogelijk om een ander GIT Project op te nemen in het huidige *working* project, zonder te hoeven schakelen tussen projecten. In de praktijk dus ideaal wanneer data van het ene project benodigd is in het andere.

Concreet zorgt een GIT Submodule ervoor dat je een andere GIT Repo als directory in je project ziet. Onderstaande commando's geven weer hoe men via de commandline zo'n GIT Submodule toevoegt.

```
> cd ~/git/myProject.git #Navigeer naar de main-GIT Repo
> ls #Bekijk de inhoud van het project
index.php
header.php
footer.php
> git submodule add ~/git/myPlugin1.git plug1
> ls #Bekijk de inhoud van het project
plug1
index.php
header.php
footer.php
> ls plug1 #Bekijk de inhoud van de toegevoegde repo
myPlugin1.js
myPlugin1.min.js
myPlugin1.css
myPlugin1_a.jpg
myPlugin1_b.jpg
README.txt
```

Hierna kan header.php uit bovenstaand myProject.git dus de stylesheet opnemen middels het pad plug1/myPlugin1.css.

Bron: (git-scm.com, Git tools - Submodules)*,* (Jean, 2009)

3.4 Advies Lectoraat

3.4.1 Status huidige situatie

Allereerst is het zaak om inzicht te krijgen in de huidige situatie. Er is gekozen om deze kennis te vergaren middels een interview aangezien dit de meest effectieve manier is om direct zaken helder te krijgen.

3.4.2 Interviewschema

Om het interview goed te laten verlopen is er vooraf een interviewschema opgesteld waarin een aantal punten zijn vastgesteld.

Agenda

- Inleiding interview (voorstellen, doel, duur, onderwerpen)
- Inleiding VCC-opdracht: doelstelling, methode en resultaten
- Uitleg & verduidelijking context zoals deze bekend is
- Interview: context vanuit opdrachtgever (schetsing en/of aanvulling)
- Interview: huidige manier van werken (hoe, wat gaat er goed en minder goed)
- Interview: wensen wat betreft. manier van werken en eisen aan bv. workflow
- Afronding: conclusie, vragen, verwachtingen

Het interview heeft plaatsgevonden op 29 Februari 2016.

3.4.3 Interviewresultaten

Naar aanleiding van het interview is de reeds bekende kennis aangevuld. In totaal maakt dit het volgende overzicht van bekende feiten.

- Er wordt reeds gebruik gemaakt van GIT
- Er is een relatief groot aantal GIT Repo's in gebruik (tussen de 10-20)
- Er wordt per project een GIT Repo gebruikt en per GIT Repo een structuur
- Er wordt vrijwel enkel gebruik gemaakt van de master branch, verder weinig tot geen branching
- Het aantal gebruikers is dynamisch, maar bedraagt altijd ten minste 4 personen
- Deze personen bevinden zich overwegend binnnen de organisatie, met uitzondering van een externe designer
- Er wordt gebruik gemaakt van JIRA en Vagrant

3.4 Vergelijking

In dit hoofdstuk zijn voor iedere workflow de voor- & nadelen op een rijtje gezet. Het verschil in de diverse workflows zit vooral in het feit dat iedere workflow geschikt is voor bepaalde situaties: kleine of juiste grote projecten, voornamelijk interne of juist veel externe gebruikers.

Centralized Workflow

+	Simpel in gebruik
+	Werkwijze vrijwel hetzelfde als SVN
-	Weinig structuur, alles in 1 ‘bak’

Feature Branch Workflow

+	Duidelijke structuur, per functionaliteit een branch
-	Geen releases

Gitflow Workflow

+	Meer controle over releases
+	Zeer geschikt bij frequente wijzigingen
-	Complexer totaaloverzicht
-	Te complex voor te kleine projecten

Forking Workflow

+	Goed toepasbaar bij niet-vertrouwde partijen
+	Geen schrijfrechten vereist voor ontwikkelaars
+	Geschikt voor grote projecten met veel externe personen
	Er is één beheerder voor de live-repository
-	Data is meer geïsoleerd
-	Complexere structuur

3.4.4 Advies

Als laatste fase in dit onderzoek zal een advies worden uitgebracht aan de opdrachtgever (in dit geval het Lectoraat) wat betreft een geschikte werkwijze wat betreft GIT.

In dit hoofdstuk zullen aan de hand van de eerder verzamelde resultaten alle gevonden GIT Workflows kort worden aangedaan met een beschrijving waarom deze wel of niet geschikt zijn voor deze situatie.

Centralized Workflow

De grootste valkuil van de Centralized Workflow is de structuur: deze is er namelijk praktisch niet, aangezien alle code in de centrale repository in 1 master branch is ondergebracht. Hierdoor wordt de branching functionaliteit van GIT niet benut.

Feature Branch Workflow

De Feature Branch Workflow is een stuk geschikter dan zijn voorganger. Doordat er per feature een branch wordt gehanteerd is er een duidelijke structuur aan te wijzen en kan er ongestoord worden gewerkt aan afzonderlijke functionaliteit.

Gitflow Workflow

Deze workflow is een uitbereiding, naast het feit dat per feature een branch wordt gehanteerd wordt bij deze workflow ook gebruikt gemaakt van een master & develop branch op het hoogste niveau. Alle features worden eerst in de develop branch opgenomen alvorens ze naar master gaan.
Dit principe maakt het beheren van releases een stuk eenvoudiger.

Forking Workflow

Deze workflow is fundamenteel anders dan eerdergenoemde. Deze workflow zorgt ervoor dat de ontwikkelaars niet direct met elkaars code verbonden zijn: enkel de beheerder kan code opnemen in de centrale repository, waarna het weer bij de ontwikkelaars terecht komt.
Ook geschikt voor situaties waarbij er gewerkt wordt met derde partijen welke men niet volledig vertrouwd, of om welke reden dan ook geen volledige toegang wilt geven.

Advies

Door de huidige & gewenste werkwijze van het Lectoraat kritisch af te leggen tegen de onderzochte GIT Workflows ben ik tot onderstaand advies gekomen, een aangepaste workflow welke delen van de behandelde workflows bevat.

Kijkend naar de digitale omgeving bij het Lectoraat is bekend dat de betrokken medewerkers overwegend intern zijn, met één externe designer. Verder zijn er een groot aantal losse GIT Repositories, aangezien er meerdere projecten lopen en er overwegend per project een repository wordt aangehouden.

Als oplossing is gekozen voor een aangepast workflow, in de kern gebaseerd op een tweetal behandelde workflows en met als toevoeging GIT Submodules. Hiermee wordt gepoogd de structuur te verduidelijken door één centrale repository te hanteren waarin de andere repositories dynamisch zijn opgenomen. Hierdoor is het wanneer gewenst mogelijk om per sub-project te werken, maar langs de andere kant alsnog een volledig overzicht te hebben door het centrale project te bekijken.

Eén centrale GIT Repository

In het middelpunt zal er één centrale repository worden aangehouden waarin het hoofdproject wordt ondergebracht, wat betreft structuur is hier gekozen voor de Gitflow Workflow. Dat houdt in dat er per grotere wijziging (of bijvoorbeeld JIRA-Ticket) een eigen branch wordt aangemaakt.

Zodra een aantal wijzigingen gereed zijn voor uitrol worden deze naar de develop branch gepusht, van waaruit vervolgens een release branch kan worden gemaakt. Vervolgens kan deze na definitieve goedkeuring naar de master worden uitgerold.

Individuele repositories voor plugins

Tijdens het onderzoek is naar voren gekomen dat er tussen de 10 en 20 repositories gebruikt worden, waarvan de meesten fungeren als opslagplaats voor extensies.

In de huidige situatie is er voor een hoop plugins een eigen repository, in de advies workflow zal deze structuur intact blijven en zullen de repositories per plugin middels GIT Submodules worden opgenomen in de centrale repository.

Voor- & Nadelen

Wanneer de geadviseerde workflow wordt geaccepteerd en geïmplementeerd vormt zich hier direct een mogelijk knelpunt: aangezien het gaat om een aangepaste workflow is het zaak dat deze eenmalig wordt geïmplementeerd en later door alle gebruikers (strict) wordt gehanteerd. Een workflow werkt immers alleen wanneer iedereen zich eraan houdt.

Ondanks dat deze workflow poogt meer overzicht & structuur aan te brengen zijn er alsnog een groot aantal repositories. Langs de ene kant is dit onoverzichtelijk, maar anderzijds biedt dit ook een bepaalde structuur.

Alternatief

Voor de centrale GIT Repository is gekozen voor de Gitflow Workflow als structuur. In plaats daarvan kan ook voor een iets minder uitgebreid alternatief worden gekozen: de Feature Branch Workflow.

In de kern is deze structuur hetzelfde, enkel zal functionaliteit vanuit de specifieke branch per functionaliteit direct worden uitgerold naar de master branch. In de praktijk zal dit als voordeel hebben dat er vlugger kan worden uitgerold, aangezien een release kan worden overgeslagen.

4. Discussie

In dit hoofdstuk zal er kritisch worden gekeken naar een aantal zaken welke invloed hebben gehad op de onderzoeksresultaten. Door deze zaken anders aan te pakken waren de onderzoeksresultaten vanzelfsprekend ook anders geweest.

4.1 Basisprincipes GIT

In paragraaf 3.1 en 3.2 ben ik ingegaan op een aantal basisprincipes van GIT en het in de praktijk brengen hiervan. Ik heb hier op internet onderzoek gedaan door een aantal handleidingen te bekijken wat betreft het gebruik van GIT. Door meerdere variaties te bekijken zijn de kernprincipes heel duidelijk naar voren gekomen, op deze manier heb ik geprobeerd zo objectief mogelijk de inhoud van dit hoofdstuk te vullen.

4.2 Veelgebruikte GIT Workflows

Door het onderzoek heen, maar vooral in hoofdstuk 3.3 worden aantal GIT Workflows behandeld welke veelgebruikt worden. Deze lijst is tot stand gekomen als resultaat van onderzoek op internet en navraag bij collega's en bekenden.

Nadat ik bij een aantal collega's en bekenden navraag heb gedaan over bij hun bekende (en gebruikte) GIT Workflows heb ik mijn zoektocht vervolgd op het internet. Hier ben ik deels dezelfde workflows tegengekomen en deels nieuwe workflows, of dezelfde met een andere naamgeving. Op basis hiervan ben ik dieper ingegaan op de juiste naamgeving en heb ik uiteindelijk een selectie gemaakt.

5. Conclusie

In dit hoofdstuk zal een sluitende conclusie worden gemaakt wat betreft het hele onderzoek. In de kern zal hier een antwoord worden gegeven op de in het begin opgestelde centrale vraag & deelvragen.

Centrale vraag

Op welke wijze kan de werkwijze (wat betreft gezamenlijk ontwikkelen) van het ontwikkelteam binnen het Lectoraat worden geoptimaliseerd, kijkend naar specifiek GIT als versiebeheersysteem?

Deelvragen

1. De basisprincipes van het versiebeheerprogramma GIT onderzoeken & begrijpen door te experimenteren zodat ik spoedig in staat ben GIT te implementeren en gebruiken in een nieuw project
2. Kennis opdoen van de diverse manieren waarop GIT kan worden gebruikt – middels zogenaamde GIT Workflows – door deze te implementeren in een testproject.
3. Onderzoek doen naar de huidige werkwijze binnen het Lectoraat door interviews af te nemen met de betrokken personen, daarnaast de huidige rol van GIT hierin herkennen en benoemen
4. In kaart brengen wat een gewenste werkwijze is en welke GIT Workflow hier het best bij past, of indien nodig zelf een aangepaste GIT Workflow definiëren

Beantwoording

De bewijslast voor de deelvragen kunnen worden teruggevonden in hoofdstuk 3.

De bewijslast voor de centrale vraag is opgeleverd in de vorm van een advies, dat kan worden teruggevonden in hoofdstuk 3, paragraf 3.3.4.

Aanbeveling

Naar aanleiding van dit onderzoek is mijn aanbeveling om het gegeven advies in hoofdstuk 3, paragraaf 3.3.4 op te volgen en de hier aangegeven werkwijze te implementeren in de dagelijkse workflow. Op deze manier kan er slimmer en meer efficiënter met GIT worden gewerkt.

Bibliografie

Atlassian. (sd). *Atlassian GIT Tutorials*. Opgeroepen op 05 16, 2016, van Atlassian.com: https://www.atlassian.com/git/tutorials/

Atlassian. (sd). *Comparing workflows.* Opgeroepen op 05 16, 2016, van Atlassian.com: https://www.atlassian.com/git/tutorials/comparing-workflows

Driessen, V. (2010, 01 05). *A successful Git branching model.* Opgeroepen op 05 16, 2016, van Thoughts and writings by Vincent Driessen.: http://nvie.com/posts/a-successful-git-branching-model/

Dudler, R. (sd). *Git - the simple guide*. Opgeroepen op 05 16, 2016, van rogerdudler.github.io: http://rogerdudler.github.io/git-guide/

Git Workflows. (sd). Opgeroepen op 05 16, 2016, van atlassian.com: https://www.atlassian.com/zh/git/workflows#!workflow-forking

git-scm.com. (sd). *Git - documentation*. Opgeroepen op 05 16, 2016, van git-sm.com: https://git-scm.com/doc

git-scm.com. (sd). *Git tools - Submodules.* Opgeroepen op 05 16, 2016, van GIT - documentation: https://git-scm.com/book/nl/v1/Git-tools-Submodules

Hodson, R. (sd). *Centralized Workflows - Ry's Git Tutorial.* Opgeroepen op 05 16, 2016, van RyPress - Quality Software Tutorials: http://rypress.com/tutorials/git/centralized-workflows

Jean, C. (2009, 06 10). *Chris Jean.* Opgeroepen op 05 16, 2016, van Git Submodules: Adding, Using, Removing, Updating: https://chrisjean.com/git-submodules-adding-using-removing-and-updating/

Bijlagen

Bijlage 1 - Zelfbeoordeling Leerdoel #1

STARRT FORMULIER	
Opleiding: HBO-ICT	
Student: Marco Verbist, **Studentnummer**: 00062951	
Docent: Wouter Everse	
S	De situatie: voor het eerste leerdoel was het zaak mij te verdiepen in de basisprincipes van GIT. Door onderzoek te doen en de ware essentie van GIT helder te krijgen verkrijg ik een dusdanige kennis van GIT dat ik later in staat ben een aantal workflows te onderscheiden, GIT in de praktijk te brengen en uiteindelijk een advies te geven aan het Lectoraat, wat betreft het gebruik van GIT.
T	Mijn taak was om onderzoek te doen naar GIT, wat het is, waar het voor staat en hoe het te gebruiken is in de praktijk.
A	Voor dit leerdoel heb ik de volgende acties ondernomen: • Onderzoek gedaan naar GIT: Wat is het? Wat zijn kernprincipes? Hoe werkt het? • Met GIT gewerkt middels testprojecten Ik ben gestart met het zoeken naar de essentie van GIT. Door op internet te zoeken naar definities van GIT werd dit al snel duidelijk. Hierna heb ik me verder verdiept in de commando's welke gerelateerd zijn aan GIT, hier ben ik achter gekomen door op internet diverse tutorials door te nemen. Iedere tutorial is uiteraard anders, maar door er meerdere door te nemen komt de essentie al gauw naar boven. Toen de basis eenmaal duidelijk was ben ik verdergegaan met het praktiseren van de tot nu toe opgedane kennis. Dit heb ik gedaan door simpelweg een project op te zetten en aan de slag te gaan met GIT. In dit geval ging het om het initiëren van een GIT Repository, het uitvoeren van commits en het mergen van branches, als mede het aanmaken van branches.
R	De resultaten van dit leerdoel heb ik verwerkt in Hoofdstuk 3.1 en 3.2. • 3.1: Basisprincipes GIT (basiscommando's, principes) • 3.2: GIT in de praktijk (uitwerking van het werken met GIT middels test-projecten)
R	Toen ik op het punt stond te beginnen met dit leerdoel had ik minimale basiskennis van GIT. Ik wist in grote lijnen wat GIT inhield en had er weleens mee gewerkt, maar de details waren mij verder onbekend. Tijdens het onderzoeken van de basisprincipes van GIT is het mij veel duidelijker geworden wat GIT is en vooral hoe je er mee werkt. Reflecterend op de concrete resultaten vind ik dat de resultaten op een goede manier tot stand zijn gekomen. Door dat ik telkens meerdere bronnen heb geraadpleegd heb ik de ware essentie (hetgeen dat telkens hetzelfde is) eruit kunnen filteren waardoor ik er zeker van kan zijn dat deze zaken juist zijn. Het direct in de praktijk brengen van de opgedane kennis heeft mij zeer veel geholpen met het bekend worden met GIT. Ik vind het zelf belangrijk om naast het 'lezen' van kennis over GIT dit ook direct in de praktijk te kunnen brengen, om zo gevoel te creëren en er mee bekend te worden. Doordat ik dit direct heb gedaan is alles mij zeer snel duidelijk geworden. Tijdens het in de praktijk brengen waren sommige zaken mij niet direct duidelijk, in deze gevallen heb ik direct opgezocht hoe de betreffende functie werkte en/of waar een bepaalde parameter voor stond. Denkend over zaken die ik anders zou aanpakken komt er niet direct iets naar boven. Ik ben heel tevreden over de 'theorie + praktijk' aanpak en zou dit zeker opnieuw toepassen, zeker ook doordat ik in het praktische gedeelte direct kan terugpakken op de theorie wanneer dit nodig is.
T	De 'theorie + praktijk' aanpak werkt voor mij heel goed, dit is daarom ook meteen een ding dat ik in de toekomst in andere situaties ook zou toepassen. De opgedane kennis tijdens dit leerdoel is tevens ook goed toepasbaar in andere situaties, zowel in de rest van dit onderzoek als buiten dit onderzoek.

Bijlage 2 - Zelfbeoordeling Leerdoel #2

STARRT FORMULIER	
Opleiding: HBO-ICT	
Student: Marco Verbist, **Studentnummer**: 00062951	
Docent: Wouter Everse	
S	Voor het tweede leerdoel was de situatie als volgt: GIT kan op diverse manieren gebruikt worden. Nu de basis van GIT bekend is, is het zaak te onderzoeken wat deze manieren zijn. Kortom: een aantal GIT Workflows onderscheiden en ze stuk voor stuk onderzoeken door er dieper op in te gaan.
T	Mijn taak was om te onderzoeken welke GIT Workflows hedendaags 'bestaan', en waar ze concreet voor staan. Zodra dit bekend is kon ik verder gaan door ze individueel te onderzoeken.
A	Voor dit leerdoel heb ik de volgende acties ondernomen: • Onderzoek gedaan naar de GIT Workflows welke veel gebruikt worden • Deze GIT Workflows verder onderzoeken: wat is de essentie en waarom zou je ze gebruiken? • Deze GIT Workflows in de praktijk brengen Ik ben gestart met het onderzoek doen naar GIT Workflows welke veel gebruikt worden. Ik heb hiervoor op internet onderzoek gedaan. Omdat ik op een aantal sites dezelfde workflows tegenkwam heb ik deze *gescoped* en hier vervolgens dieper onderzoek naar gedaan door ze stuk voor stuk te bekijken. Onderdeel van dit stuk voor stuk bekijken was om de workflows allemaal in de praktijk te brengen: dit heb ik gedaan door telkens een klein projectje te starten, gebruikmakend van de betreffende workflow. Hierbij heb ik telkens de resultaten gedocumenteerd en opgenomen in de resultaten.
R	De resultaten van dit leerdoel zijn te vinden in hoofdstuk 3.3 en Bijlage 5. • 3.3: GIT Workflows • 5.3 t/m 5.6: Diverse workflows in de praktijk gebracht
R	Voor ik aan deze deelvraag begon had ik al basiskennis van GIT maar bestond mijn 'workflow' puur uit het maken, aanpassen en 'committen' van bestanden. Van branching had ik gehoord, maar ik had het nooit in de praktijk gebracht. Voor de eerste activiteit van dit leerdoel heb ik op internet onderzoek gedaan naar een aantal workflows. Bij dit onderzoek kwam ik telkens terug op ongeveer vier dezelfde workflows. Toen bleek dat deze 4 op meerdere sites stonden uitgewerkt was mij duidelijk dat deze vier veel gebruikt worden. Om hier zeker van te zijn heb ik verder gezocht: ik ben blijven zoeken en heb gesproken met onder andere klasgenoten en collega's om dit te bevestigen. Toen bleek dat hun omschrijving van de workflows overeenkwam en hun met vrijwel dezelfde workflows kwamen was ik eruit. Ik heb hier verder onderzoek gedaan omdat ik zeker wilde weten dat de workflows welke ik had gevonden de juiste waren, en ik niet op een dwaalspoor zat. Tijdens het uitwerken van de workflows heb ik regelmatig op internet gezocht hoe iets ook al weer zat, of waar iets voor stond. Achteraf een goed zet, aangezien alle 'kleine' toevoegingen/parameters mij nu helemaal duidelijk zijn. Had ik dit niet gedaan dan was mijn kennis een stuk beperkter gebleven. Terugkijkend op deze deelvraag ben ik tevreden met mijn aanpak, hoewel er wel wat puntjes van aandacht zijn. Na de initiële feedback van begeleidend docent Wouter Everse was duidelijk dat mijn uitwerking wat uitgebreider mocht. Naar aanleiding hiervan ben ik dieper aan doorpakken en heb ik voornamelijk deze deelvraag wat uitgebreider gemaakt, waardoor er nu per workflow meer uitleg is. In een toekomstig soortgelijke situatie zou ik dit beter vanaf het begin uitgebreider kunnen aanpakken.

T	In de vorige deelvraag ging het om basiskennis van GIT, in deze deelvraag gaat het om kennis wat betreft de manier van werken met GIT. Basiskennis is vereist, maar aanvullende kennis is altijd mooi meegenomen. Nu ik deze kennis ook heb kan ik nóg beter met GIT werken, in elke situatie waar GIT gebruikt wordt. Wat betreft aanpak zou ik in een volgende soortgelijke situatie direct dieper doorpakken in plaats van middels meerdere sessies (en feedbackmomenten) tot het resultaat te komen.

Bijlage 3 - Zelfbeoordeling Leerdoel #3

STARRT FORMULIER	
Opleiding: HBO-ICT	
Student: Marco Verbist, **Studentnummer**: 00062951	
Docent: Wouter Everse	
S	Voor het derde leerdoel was de situatie dat ik duidelijkheid wou krijgen wat betreft de huidige werkwijze met GIT binnen het Lectoraat. Concreet was het doel om duidelijkheid te krijgen over de manier van werken en daar vervolgens (in de laatste deelvraag) een advies over uit te brengen.
T	Om duidelijkheid te krijgen heb ik besloten om een interview af te nemen. Mijn taak was daarom om in eerste instantie uit te zoeken bij wie ik dit interview kon afnemen, als tweede een aantal interviewvragen op te stellen, als derde het interview uit te voeren en als laatste de resultaten te verwerken.
A	Voor dit leerdoel heb ik de volgende acties ondernomen: • De kennis die ik al had op een rijtje gezet • Een aantal punten genoteerd waar ik antwoord op wilde hebben • Een interview afgenomen • De resultaten verwerkt Voor dit leerdoel ben ik gestart met het 'verzamelen' van de kennis die ik al had, niet veel in dit geval. Hierna heb ik mezelf heel kritisch de vraag gesteld: wat wil ik weten van het Lectoraat? Op basis hiervan heb ik een aantal punten genoteerd welke ik wil meenemen in het interview. Daarna heb ik een afspraak gemaakt en een interview afgenomen. Na afloop van het interview heb ik de vergaarde kennis verwerkt zodat ik dit in een later stadium als basis kon dienen.
R	De resultaten van dit leerdoel zijn terug te vinden in hoofdstuk 3.4. • 3.4.1: Wat er bekend is over de huidige situatie • 3.4.2 Interviewschema • 3.4.3 Interviewresultaten (verwerkt)
R	Voor deze deelvraag heb ik gekozen voor een interview omdat me dit de meeste logische manier leek om meer informatie te verkrijgen over de huidige werkwijze wat betreft GIT. Het stappenplan waarbij ik eerst bestaande informatie op een rij zet, daarna een aantal interviewpunten noteer, het interview afneem en tot slot de resultaten verwerk leek me geschikt omdat dit de manier is waarop ik al vaker interviews heb afgenomen. Deze manier blijkt in de praktijk altijd goed te werken. Tijdens het interview was mijn insteek om zoveel mogelijk informatie te verkrijgen, zowel over de huidige werkwijze als over de gewenste werkwijze. Terugkijkend naar hoe dit is gegaan ben ik hier erg tevreden over.
T	Waar het in de vorige leerdoelen ging om 'kennis' van GIT gaat het in deze deelvraag om de huidige werkwijze van het Lectoraat. Deze opgedane kennis is buiten het onderzoek niet heel relevant en kan daarom alleen gebruikt worden in dit onderzoek. Later in dit onderzoek om precies te zijn, op het punt waar de verbeterpunten liggen. De werkwijze wat betreft het interview is na afloop wederom geschikt bevonden, in een volgend interview zou ik deze methode daarom aanhouden.

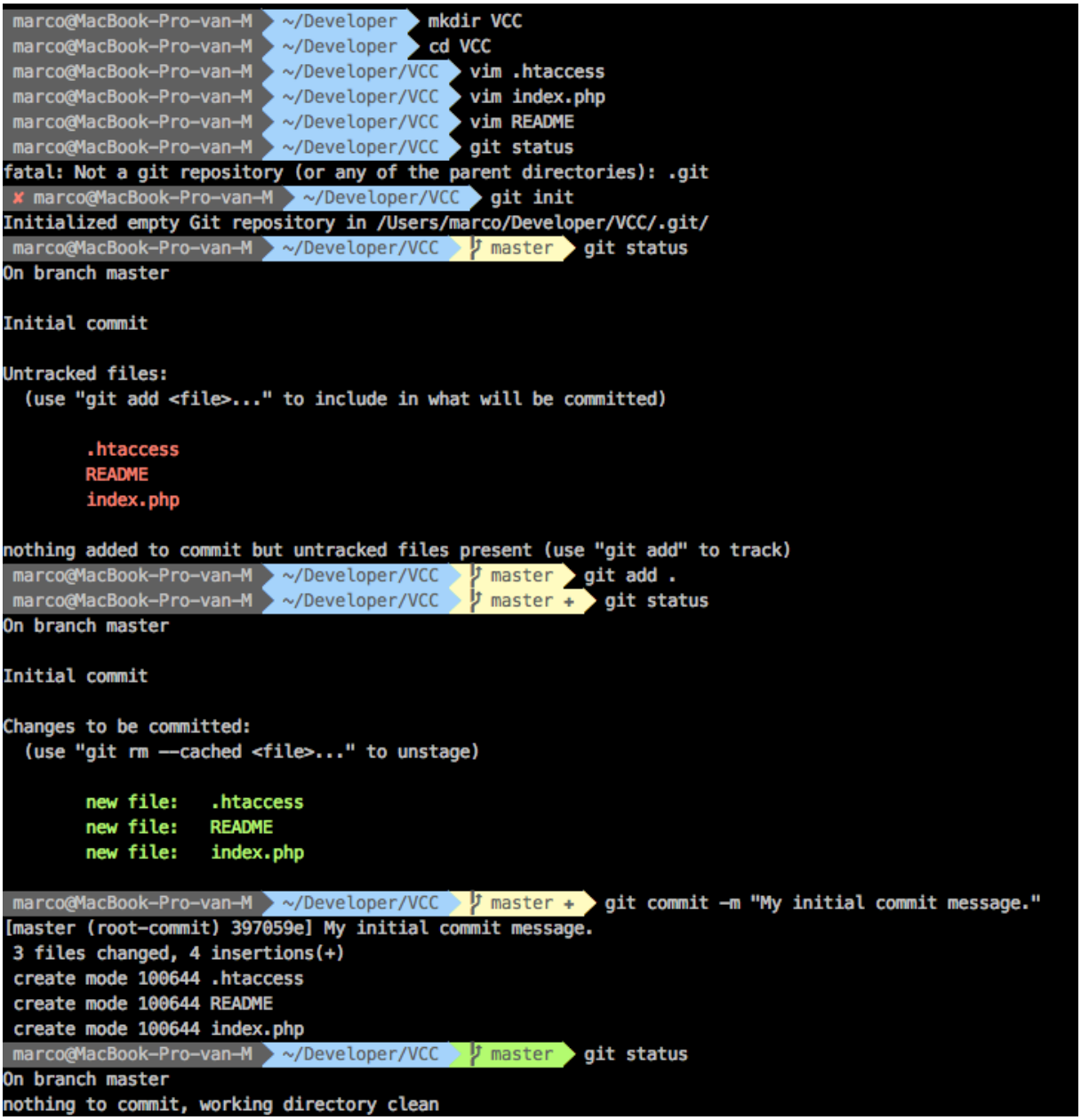
Bijlage 4 - Zelfbeoordeling Leerdoel #4

STARRT FORMULIER	
Opleiding: HBO-ICT	
Student: Marco Verbist, **Studentnummer**: 00062951	
Docent: Wouter Everse	
S	Voor het vierde en laatste leerdoel was de situatie als volgt: breng een advies uit richting het Lectoraat wat betreft een verbeterde werkwijze voor het gebruik van GIT.
T	Nadat ik in eerdere deelvragen randzaken had uitgezocht was het in deze deelvraag zaak om een advies uit te brengen richting het Lectoraat, wat betreft de manier van werken met GIT.
A	Voor dit leerdoel heb ik de volgende acties ondernomen: • De huidige werkwijze bekeken • Bedacht uit welke puzzelstukjes de gewenste werkwijze zou kunnen bestaan • Een advies opgesteld • Voor- en Nadelen bepaald en een alternatief opgesteld Ik ben dit leerdoel gestart door de huidige werkwijze te bekijken, en te kijken in hoeverre hier gebruikt wordt gemaakt van de functionaliteiten welke GIT biedt. Hierna heb ik voor de onderzochte workflows bekeken in hoeverre deze passend zijn voor de casus. Uiteindelijk heb ik een advies opgesteld waarin wordt beschreven welke aangepaste workflow geschikt is, in dit advies is naast de voor- en nadelen van dit advies ook een alternatief beschreven.
R	Het resultaat van dit leerdoel is terug te vinden in hoofdstuk 3.4, paragraaf 3.4.4 voor het advies.
R	Wat betreft de reflectie is deze deelvraag het meest interessant. Doordat ik in een eerdere deelvraag de gevonden workflows heb uitgewerkt was mij bekend hoe deze workflows in de praktijk werkend. Door deze kennis naast de situatie bij het Lectoraat te houden kon ik zo bepalen welke samenstelling geschikt is. Tijdens dit proces heb ik meerdere situaties geschetst, aan de hand van de eisen & wensen van het Lectoraat heb ik uiteindelijk een oplossing geschetst, dit is een combinatie van twee workflows inclusief enkele aanpassingen geworden. Nadat ik deze aangepaste workflow (het advies) had geschreven heb ik hier kritisch op teruggeblikt: voldoet het aan de wensen? Is het bruikbaar? Toen ik een eerste conceptversie van mijn onderzoeksportfolio naar begeleidend docent Wouter Everse had gestuurd kreeg ik als feedback op het advies dat de koppeling met de context niet helder was. Naar aanleiding van deze feedback heb ik het advies uitgebreider gemaakt, iets wat ik achteraf gezien al eerder had kunnen doen door het advies kritischer door te lezen.
T	In een volgende situatie zou ik een advies kunnen verbeteren door een kritischer op terug te kijken. Na het eerste concept ben ik zelf te vlug tevreden geweest met het advies. De herbruikbaarheid van het resultaat van dit leerdoel ligt in het advies. Op basis van mijn advies kan het Lectoraat (wanneer er akkoord wordt gegaan) de workflow aannemen en gaan gebruiken. Voor andere organisaties zal deze workflow in de huidige vorm niet geheel bruikbaar zijn, aangezien een workflow uiteindelijk iets heel specifieks is dat vaak op maat wordt gemaakt voor iedere organisatie, aangezien iedere organisatie andere behoeften heeft.

Bijlage 5 - GIT Praktisering

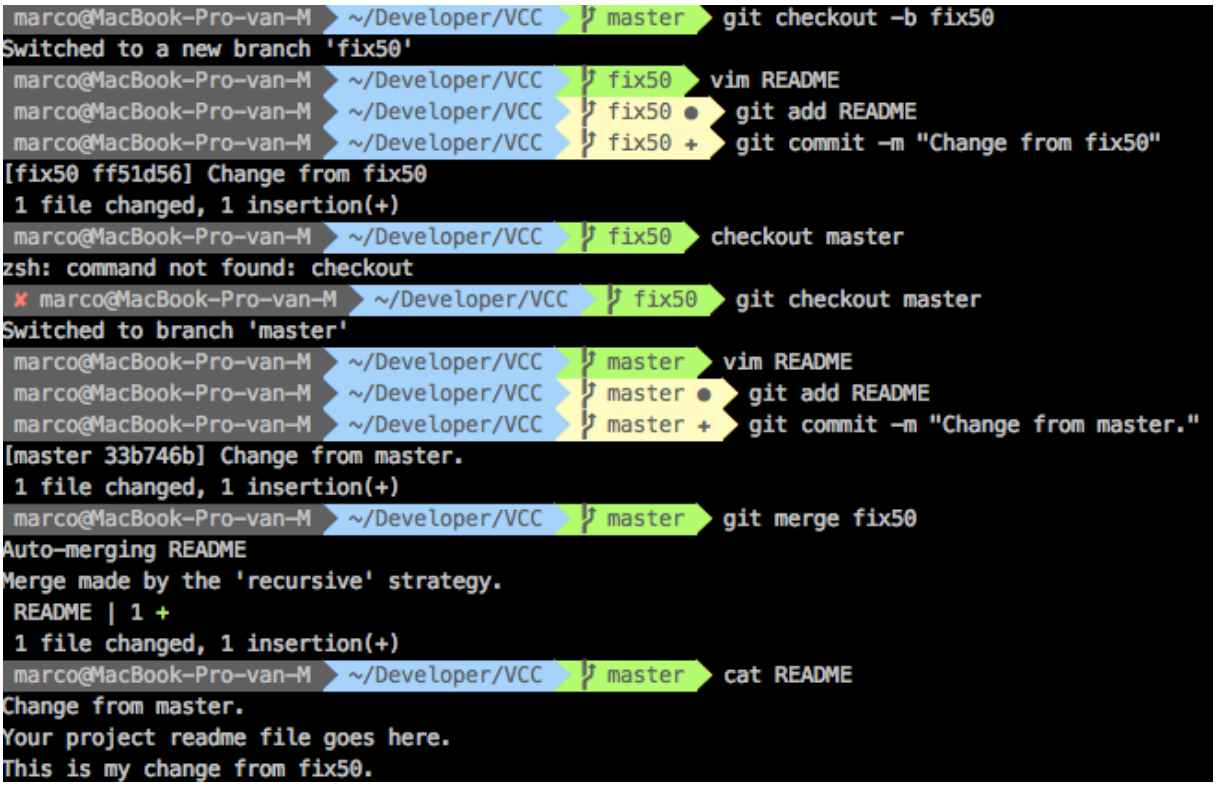
5.1 Basis GIT

In onderstaande afbeelding is te zien hoe de basisprincipes van GIT in de praktijk worden gebracht. Er wordt een lege GIT Repository geïnitieerd waarin vervolgens enkele basisacties worden uitgevoerd.



5.2 Merging

In onderstaande afbeelding is te zien hoe gegevens van twee branches (fix50 en master) worden samengevoegd.



5.3 Centralized Workflow

Bij de Centralized Workflow begint de ontwikkelaar door de bestaande repository lokaal over te nemen, middels het `git clone` commando.

```
> git clone http://externalServer.com/VCC.repo
> cd VCC
> ls -toha
drwxr-xr-x  14 marco   476B 10 mei 09:50 .git
drwxr-xr-x   7 marco   238B 10 mei 09:50 ..
drwxr-xr-x   7 marco   238B  2 feb 15:17 .
-rw-r--r--   1 marco    86B  2 feb 15:17 README
-rw-r--r--   1 marco    85B  2 feb 14:59 index.php
-rw-r--r--   1 marco    23B  2 feb 14:37 header.php
-rw-r--r--   1 marco    29B  2 feb 13:45 .htaccess
```

Bovenstaande code laat zien dat er vervolgens een VCC-directory beschikbaar is met daarin een aantal bestaande bestanden. De vervolgstap is dat de gebruiker een aantal wijzigingen maakt, en lokaal commit. Dit mag gewoon in de standaard master branch.

```
> nano header.php // Open file, make changes, save & close
> nano index.php // Open file, make changes, save & close
> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

      modified:   header.php
      modified:   index.php

no changes added to commit (use "git add" and/or "git commit -a")
> git commit –a // Typ some Commit Description
[master 12d70e7] My changes..
 2 files changed, 2 insertions(+), 1 deletion(-)
```

De wijzigingen in de twee aangepaste bestanden zijn nu opgeslagen in de lokale GIT Repository. Als laatste stap in de Centralized Workflow is het zaak om de lokale repo, te pushen naar de server.

```
> git push origin master
```

Met dit commando worden alle lokale wijzigingen sinds de laatste clone terug gepusht naar de server. Indien een andere persoon eerder zijn wijzigingen heeft terug gepusht zal deze actie falen, het is dan zaak dat de lokale GIT Repository eerst wordt gesynchroniseerd voordat er gepusht kan worden.

```
> git pull --rebase origin master
> git status
# Unmerged paths:
# (use "git reset HEAD <some-file>..." to unstage)
# (use "git add/rm <some-file>..." as appropriate to mark resolution)
#
# both modified: header.php
# both modified: index.php
```

Het bovenstaande `git status` commando geeft vervolgens een overzicht van bestanden waarin zogenoemde `merge conflicts` voorkomen. Het is nu zaak dat voor ieder bestand waarin een conflict aanwezig is, dit verholpen wordt. Hierna kan in GIT worden aangegeven dat het conflict is opgelost middels het volgende commando.

```
> git add header.php
> git add index.php
> git rebase --continue
```

Het `git rebase` commando zorgt ervoor dat GIT verdergaat met het samenvoegen. Wanneer dit proces voltooid is kan het commando waar het allemaal mee begon weer worden uitgevoerd:

```
> git push origin master
```

De lokale wijzigingen zullen nu zijn gepusht naar de centrale repo, waarbij rekening is gehouden met wijzigingen welke gemaakt zijn nádat de repo in eerste instantie was overgenomen.

5.4 Feature Branch Workflow

Wanneer er in een team gebruik wordt gemaakt van de Feature Branch Workflow is het belangrijkste principe dat er per feature een branch wordt gehanteerd. In eerste instantie zal er daarom lokaal een nieuwe branch worden aangemaakt wanneer er gestart wordt met nieuwe functionaliteit.

```
> git checkout –b wysiwyg-editor
> git status
On branch wysiwyg-editor
nothing to commit, working directory clean
```

Aangezien het hier om een nog niet bestaande branch gaat wordt de –b vlag toegevoegd, de branch zal nu worden aangemaakt en direct worden ingesteld als actieve branch, working branch. Het `git status` commando geeft vervolgens aan dat de branch leeg is, zoals te verwachten was.

Wanneer er vervolgens een aantal wijzigingen zijn gemaakt kunnen deze in GIT worden verwerkt.

```
> git status
On branch wysiwyg-editor
Untracked files:
  (use "git add <file>..." to include in what will be committed)

      editor.js
      style.css

nothing added to commit but untracked files present (use "git add" to
track)
> git add editor.js
> git add style.css
> git commit –a // Typ a commit description
```

De wijzigingen zijn nu helemaal opgenomen in de lokale GIT Repository, in de `wysiwyg-editor` branch. Wanneer de functionaliteit af is kan de branch worden gepusht naar de centrale repository.

```
> git push –u origin wysiwyg-editor
```

De vlag –u origin geeft aan dat de lokale branch met de naam `wysiwyg-editor` vanaf nu is gekoppeld aan de gelijknamige branch op de centrale repository. Hierna kan simpelweg `git push` worden uitgevoerd (zonder de extra parameters) om de branch bij te werken op de centrale repository.

Features opnemen in master

Zodra het moment is aangebroken dat één of meerdere features (branches) kunnen worden opgenomen in de live versie kunnen deze worden samengevoegd met de master branch. Dit gaat als volgt.

```
> git checkout master
> git merge wysiwyg-editor
> ls –to
-rw-r--r--  1 marco   2 10 mei 10:33 editor.js
-rw-r--r--  1 marco   4 10 mei 10:33 style.css
-rw-r--r--  1 marco  86 10 mei 10:21 README
-rw-r--r--  1 marco  88 10 mei 09:55 index.php
-rw-r--r--  1 marco  24 10 mei 09:55 header.php
> git branch –d wysiwyg-editor
> git branch
  fix50
* master
```

Hier wordt eerst overgestapt naar de master branch waarna de `wysiwyg-editor` branch wordt ingevoegd in de master branch. Als laatste stap wordt de `wysiwyg-editor` branch verwijderd, aangezien deze nu niet meer nodig is.

Indien gewenst kan deze branch wel worden aangehouden voor eventuele latere doorontwikkeling van de functionaliteit.

5.5 Gitflow Workflow

Zoals eerder vermeld lijkt de Gitflow Workflow veel op de Feature Branch Workflow, het verschil hier zit 'm voornamelijk in het feit dat feature-branches bij deze workflow niet direct worden gemerged met de master branch, maar eerst in een developer branch.
Een voorbeeld:

```
> git branch develop
> git push –u origin develop

> git checkout –b feature1
// Work with some files in this branch and commit them
> git checkout –b feature2
// Work with some files in this branch and commit them
```

In bovenstaande code wordt eerst lokaal een `develop` branch aangemaakt, waarna deze ook op de centrale repository wordt aangemaakt.

Vanaf dit punt kan lokaal functionaliteit worden ontwikkeld middels feature branches. In dit voorbeeld heten deze respectievelijk `feature1` en `feature2`.

```
> git pull origin develop
> git checkout develop
> git merge feature1
> git merge feature2
> git push
> git branch –d feature1
> git branch –d feature2
```

Na het ontwikkelen is het zaak dat de features worden langzamerhand omhoog worden gezet richting de release. Allereerst wordt gezorgd dat alle externe wijzigingen in de develop branch lokaal zijn overgenomen, hierna wordt de develop branch direct ingesteld als actieve branch.

Vervolgens worden de twee feature-branches feature1 & feature2 samengevoegd in de develop branch, mits er geen conflicten voorkomen uiteraard. Hierna wordt met git push de lokale develop branch bijgewerkt in de centrale repository.

Als laatste (en wederom optionele) stap worden de twee feature branches verwijderd, aangezien de in principe niet meer nodig zijn. Indien gewenst kunnen ze uiteraard ook behouden blijven, zodat er later in verder gewerkt kan worden.

Het maken van een release

Naarmate de ontwikkeling volgt zullen steeds meer feature-branches worden samengevoegd in de develop branch.

```
> git checkout –b release-1.0 develop
```

Met bovenstaand commando zal een nieuwe branch met naam ‘release-1.0’ worden gemaakt, op basis van develop. Dat betekend concreet dat alle inhoud van develop tot dit punt ook beschikbaar is in de nieuwe release-1.0 branch.

Op dit punt kan de release-1.0 branch gebruikt worden om bijvoorbeeld tests uit te voeren. Zodra dit is voltooid kan de *release* worden uitgebracht. In onderstaande code wordt de release-1.0 branch samengevoegd met de master branch. Concreet betekend dat, dat alle ontwikkelde functionaliteit nu ook daadwerkelijk live staat.

```
> git checkout master
> git merge release-1.0
> git push
```

Als laatste stap is het belangrijk dat eventuele wijzigingen in de release-1.0 branch worden teruggebracht naar de develop branch. Het kan hier bijvoorbeeld gaan om documentatie of een README-file welke gewijzigd is.

```
> git checkout develop
> git merge release-1.0
> git push
```

Hierna kan gekozen worden om de release-1.0 branch te verwijderen. Als laatste kan voor een groter gemak worden gekozen om de release te voorzien van een tag, dit kan middels het git tag commando.

```
> git tag –a 1.0 –m “First release.” master
```

Door dit commando uit te voeren wordt de huidige versie van de master branch ‘getagd’ als ‘1.0’, met de bijhorende omschrijving.

5.6 Forking Workflow

Zoals eerder vermeld is de Forking Workflow totaal anders dan de vorige behandelde workflows. Bij de Forking Workflow heeft iedere ontwikkelaar een eigen repository op de server, waardoor enkel de centrale beheerder schrijfrechten nodig heeft tot de centrale repository, en dus de master repo.

Deze structuur is handig in situaties waarbij men zuinig wilt omgaan me toegangsrechten of een bepaalde partij niet volledig vertrouwd en deze dus geen toegang tot de codebase wilt geven.

Deze workflow begint in de kern met een centrale repository, waar de master branch fungeert als middenpunt. Vanaf deze centrale repository worden meerdere forks gemaakt, een fork per ontwikkelaar. Ontwikkelaars kunnen pullen van elkaars repositories maar pushen kan enkel naar hun eigen repository.

Aangezien men bij deze workflow enerzijds zeer gebonden maar anderzijds heel vrij is wat betreft de opzet (bijvoorbeeld branching) is hieronder een mogelijke structuur uitgewerkt.

De centrale repository

Allereerst zet de beheerder een centrale repository op middels het commando git init.

```
> git init --bare /home/admin/git/central.repo
```

Repository's per ontwikkelaar

Iedere ontwikkelaar kan nu vanaf de server de centrale repo clonen naar zijn eigen kopie, ook op de server. De naam van de doelrepo maakt hierbij niet uit.

```
> git clone /home/admin/git/central.repo /home/peter/data.repo
> git clone /home/admin/git/central.repo /home/marco/data.repo
> git clone /home/admin/git/central.repo /home/ernst/mainProject.repo
```

Nadat dit gedaan is kan deze kopie lokaal worden overgenomen op de ontwikkelsystemen.

```
> git clone http://externalServer.com/peter/data.repo
```

Lokaal ontwikkelen

Vanaf dit punt kunnen de ontwikkelaars lokaal aan hun functionaliteit(en) werken, zij zijn hier in principe vrij om een eigen structuur aan te houden. Zo kunnen ze bijvoorbeeld vrijwel zonder branches werken zoals het bij de Centralized Workflow gaat, of kiezen voor (een variant op) de Feature Branch- of Gitflow Workflow.

```
> git checkout –b feature1
// Do some work in branch feature1 & commit
```

Zodra een ontwikkelaar één of meerdere functionaliteiten af heeft kan hij de bijhorende branches pushen.

```
> git push origin feature1
```

Op dit punt staan de wijzigingen van de ontwikkelaar in zijn eigen repository op de centrale server, in de feature1 branch. Zodra de beheerder op de hoogte is gebracht kan deze de wijzigingen opnemen. Dit kan of direct in de master branch, of zoals eerder beschreven kan ook hier weer een vorm van branching plaatsvinden. Zo kan er bijvoorbeeld gekozen worden om naast de master branch ook een develop branch in het leven te roepen, alle wijzigingen zullen dan eerst hierin verzameld waarna er op basis van de develop branch een release wordt gemaakt, welke vervolgens weer met live zal worden gemerged.

```
> git fetch http://externalServer.com/peter/data.repo feature1
> git merge FETCH_HEAD
```

Bovenstaande code is zeer interessant, het commando `git fetch` haalt data uit een externe repository op, in dit geval enkel de branch feature1, uit – in dit geval – de repository van een ontwikkelaar, vervolgens is deze branch beschikbaar als FETCH_HEAD.

Middels het commando `git log FETCH_HEAD` kan de commit-geschiedenis worden weergegeven, en middels `git merge` wordt alle data uit de externe feature1 branch overgenomen in de huidige working branch, bijvoorbeeld de master branch. (Afhankelijk van de gehanteerde branching-structuur)

Als laatste stap dient de beheerder de nieuwe ‘master’ branch op te slaan middels het commando `git push origin master`, wanneer de ontwikkelaars nu bijwerken hebben zij ook de nieuwste wijzigingen tot hun beschikking.

Bijlage 6 – Deelnameformulier

Student:	Marco Verbist	**Studentnr:**	00062951
SLC'er:	Wouter Everse		
Extern begeleider:	Wouter Everse	**Organisatie:**	
PVCC en code:			
EC:			

Datum beoordeling:

Handtekening voor akkoord
Extern begeleider

Toelichting

Bijlage 7 – Urenverantwoording

In onderstaande tabel zijn de beoogde uren wat betreft uitvoering van de VCC-opdracht opgenomen zoals deze bij aanvang zijn ingeschat. De tweede kolom bevat de werkelijk gemaakte uren per onderdeel.

Uren beoogd	Uren praktijk	Omschrijving
2	3	Voorbereiding VCC: vergaren opdracht, opstellen POP-PAP
23	25	Basisprincipes GIT onderzoeken & praktiseren
23	23	Kennis hebben van GIT Workflows
23	26	Onderzoek doen naar huidige werkwijze Lectoraat
+	+	
71	77	

Zoals hierboven duidelijk wordt is het initieel geschatte aantal uren niet gehaald en is er meer tijd aan besteed. De oorzaak hiervan ligt bij het feit dat de inschatting vrij laag bleek en de taken in de praktijk meer tijd nodig hadden.