HUSACCT Git repository version 2

Structuur en workflow

Auteurs: Tom de Bruijn met input van René van Aerle

Document versie: 1.1

Contact:

- Team 1 Control – beheer Git repository
  - René van Aerle (info@rvgate.nl)
  - Backup moet nog toegekend worden.
- Team 5 Architecture Graphics – Git support
  - Tom de Bruijn (tomdebruijn@live.nl)

Inhoudsopgave

De huidige opzet

De huidige Git repository structuur is een beetje “een rommeltje geworden” (Zie plaatje). De fork structuur van Git wordt nu niet goed toegepast en er is geen duidelijke parent-child relatie tussen de *“main repository”* en de fork repositories. Daarbij komt dat er geen structuur was aan het begin. De effecten daarvan kunnen nog gemerkt worden door rommelige package structuur (extra niet afgesproken packages), verschillende implementaties van Main files, etc. Dit is niet een resultaat dat komt door de opzet van de huidige Git structuur, echter dit komt door een gebrek aan project structuur bij het begin van het gebruik van Git.

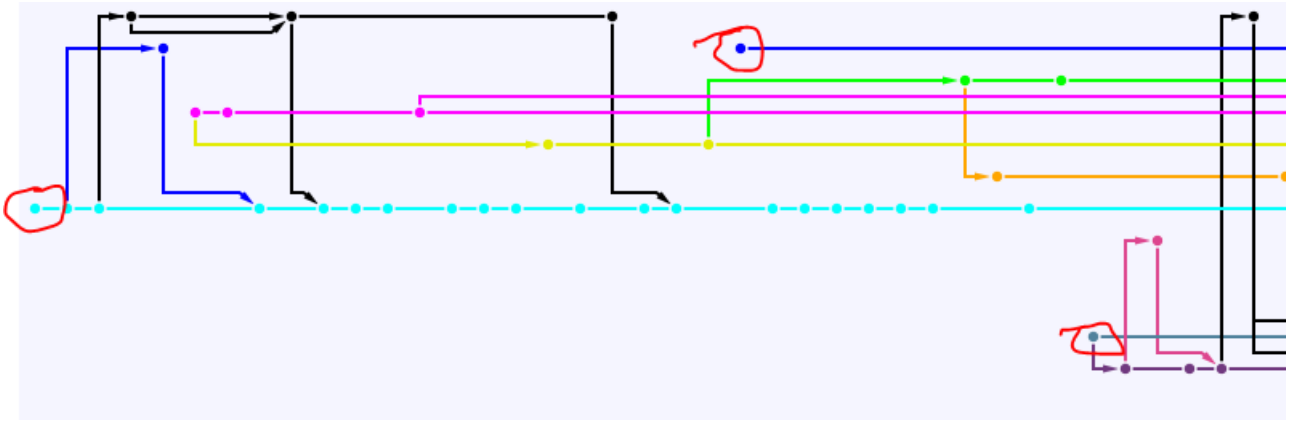


Illustration 1: Heel klein onderdeel van de huidige situatie van de main Git repository met de merge toegepast. 3 points of origin (gehighlight met rood), met nog andere 4 buiten beeld.

Het probleem is dat als iedereen op de huidige versie blijft verder bouwen bepaalde problemen zich kunnen blijven herhalen bij iedere merge. Ik zie een probleem dat veel groter gaat worden in de toekomst. Dit zal iedereen zijn beeld van Git in een slecht daglicht zetten, maar belangrijker, zal het er voor zorgen dat de merges van het project steeds tijd intensiever worden om uit te voeren. Om voortaan een goed beheerde *“main repository”* aan te bieden moet er een betere structuur komen.

Ik zeg het niet graag, maar het is beter om opnieuw te beginnen. Niet het gehele project, maar de Git repositories. Verder gaan met de code, maar een nieuwe Git structuur af te spreken. Één die nauwer samenwerkt en minder afhankelijk zou moeten zijn van de inzet van team 1. Op de huidige manier voegen de forks van de team repositories geen extra functionaliteiten of overzicht toe.

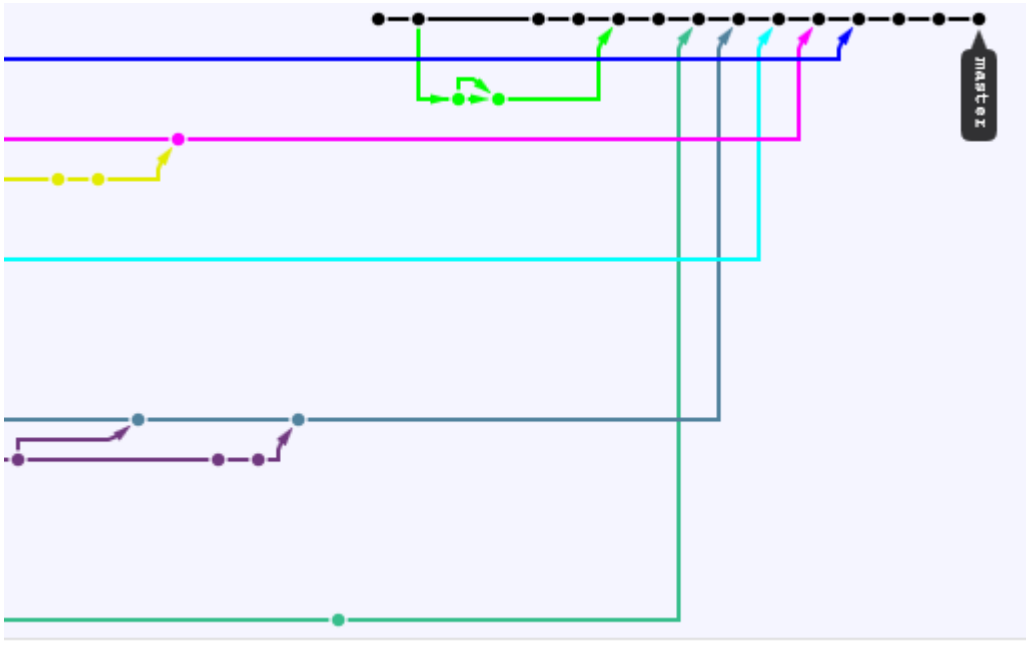


Illustration 2: Resultaat van de merge door team 1. Points of origin beginnen voor de “main repository” en zijn directe afgeleide ervan.

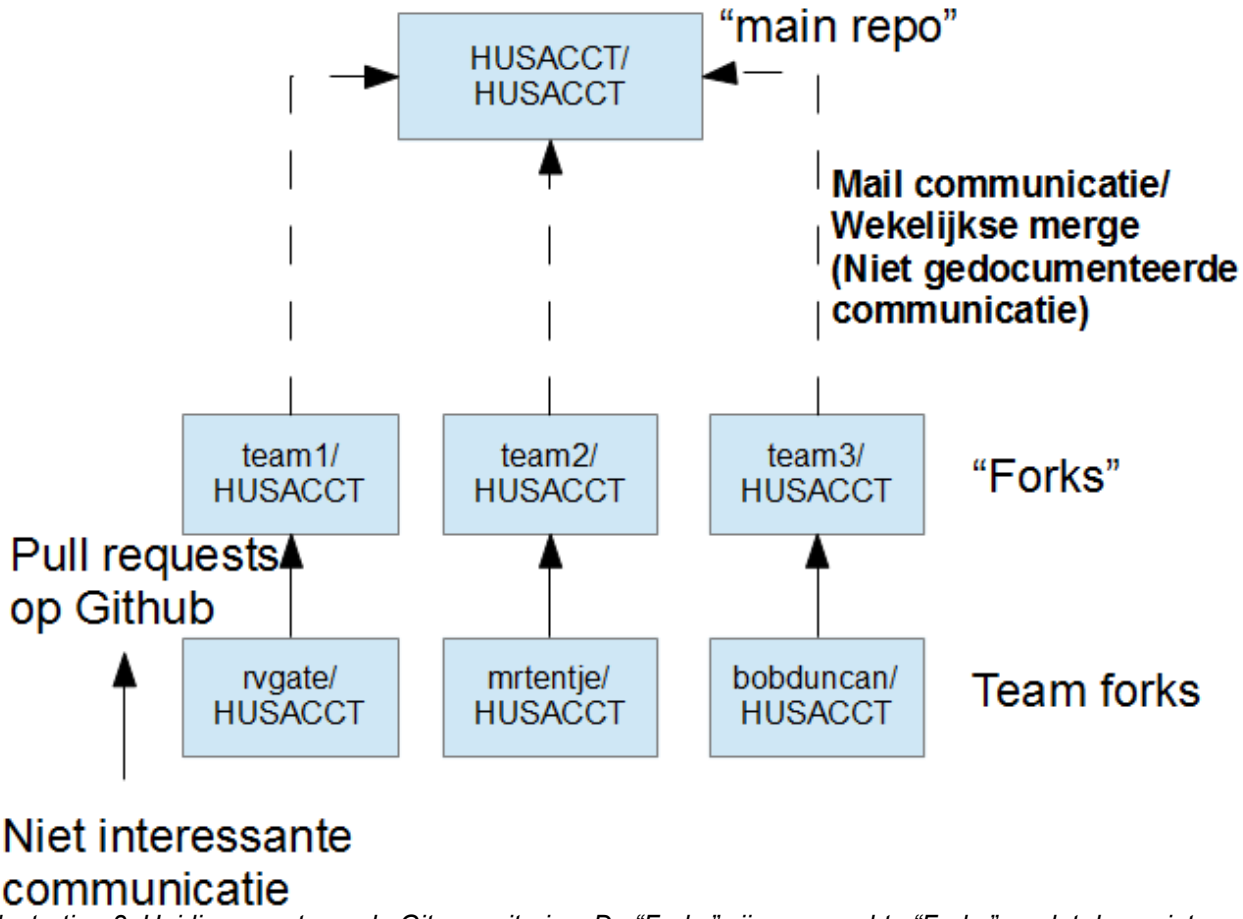


Illustration 3: Huidige opzet van de Git repositories. De “Forks” zijn geen echte “Forks” omdat deze niet afgeleid zijn uit de “main repository”.

Nieuwe opzet

De nieuwe opzet is een opzet die al eerder besproken was en gerealiseerd zou worden. Voortaan zouden de repository beheerders van ieder team een fork moeten maken van de "*main Git repository*". Er is dus voortaan een directe relatie tussen de *"main repository"* en de team repositories. Deze relatie kan op Github, gedocumenteerd op de repository pagina, terug gevonden worden.

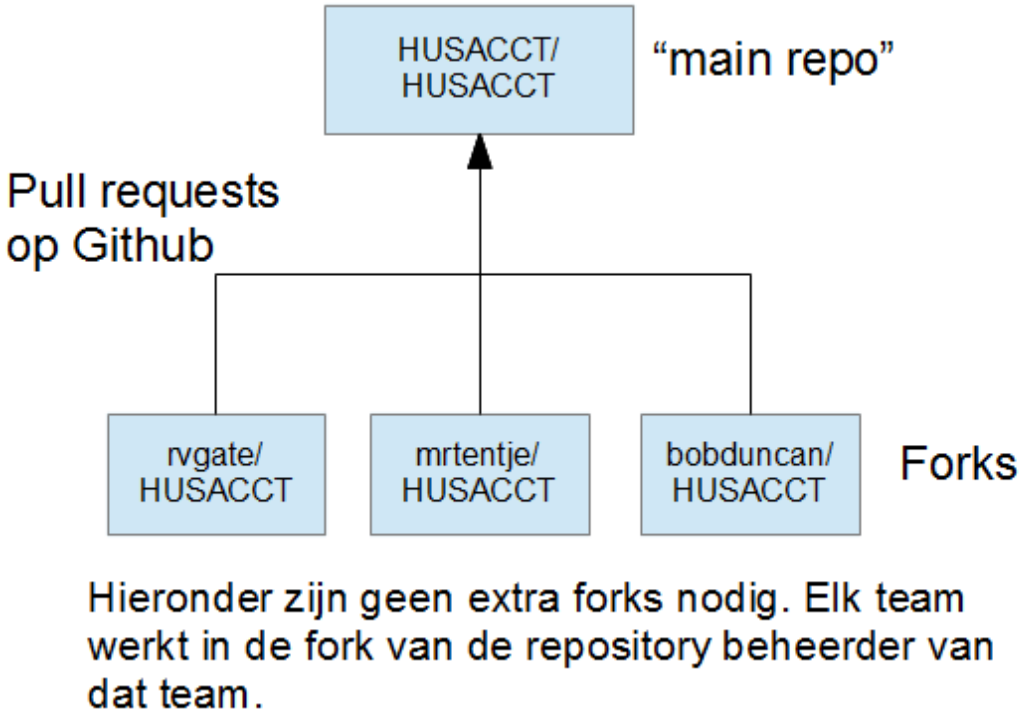


Illustration 4: Nieuwe structuur.

Communicatie bevindt zich dan volledig op Github. Dit is goed te doen; tenslotte veel open source projecten doen dit ook. Hoe dit precies in zijn werk gaat wordt beschreven bij het kopje "Werkmethode".

Voordelen

De voordelen zijn als volgt:

- Er is een **begin** van het project in de Git repository. Er zijn geen zes "points of origin" omdat het eigenlijk zes aparte repositories zijn die telkens samengevoegd worden.
- Er is een structuur gedefinieerd **vooraf** en hierin werkt ieder team **verder**.
- Er komt een beter overzicht van welke forks bestaan. Er is een duidelijke relatie aangegeven op Github, niet alleen binnen ons project.
    - Er is een parent/*"main repository"*: HUSACCT/HUSACCT
    - En er zijn child repos. Deze zijn op Github in de *"main repository"* terug te vinden.
      Forks → Members:
        - rvgate/HUSACCT
        - mrtentje/HUSACCT
        - gitteam3/HUSACCT
        - gitteam4/HUSACCT
        - Newanz/HUSACCT
        - gitteam6/HUSACCT

- Github bevat een makkelijker, geautomatiseerd, systeem voor pull requests. Er is in principe geen onderlinge mail communicatie nodig voor pull requests. Dit zal echter hoogst waarschijnlijk wel gebeuren.
    - De documentatie van iedere merge is volledig beschikbaar op Github.
        - Ieder team kan gemakkelijk op Github terug vinden welke pull requests zijn toegepast en welke nog behandeld moeten worden.
    - Het neemt één extra stap voor mergen weg, omdat Github dit zelf op probeert te lossen.
        - Dit zegt **niet** dat het **een perfect systeem** is. Er wordt dan ook geadviseerd om eerst zelf pull requests op de lokale versie van de *"main repository"* uit te voeren voordat de pull request geaccepteerd wordt. Dit kan heel gemakkelijk door deze fork met de *"main repository"* te mergen en in Eclipse te testen. Werkt het goed dan kan de pull request toegepast worden op Github zelf. Zo niet dan kan dit terug gecommuniceerd worden aan het team in kwestie zodat zij de build issues kunnen verhelpen.
            - Voor meer informatie en een stappen plan, zie: http://help.github.com/send-pull-requests/
        - Team 1 moet dus de merge van te voren uit proberen,maar dit zal soepeler moeten verlopen omdat alle forks gebaseerd zijn op dezelfde *"main repository"*.
- Er kan alsnog onderling gemerged worden met andere teams, maar zou niet hoeven.
    - Niet zozeer omdat het volgens de "Git way" niet mag, maar omdat ik zou aanraden dit niet uit te voeren omdat dit onnodige complexiteit kan brengen in een project waar 90% van de mensen nog geen tot weinig kennis hebben van Git.
    - Onderdelen die af zijn, moeten met een pull request in de *"main repository"* beschikbaar komen. Daarna kunnen andere teams daarvan weer de wijzigingen binnen halen. Code wordt dus gedeeld vanuit één locatie i.p.v. 7.
- Er kan veel vaker gemerged worden via pull requests, niet alleen eens per week.
    - Deze merges zijn gedocumenteerd (op Github) en zichtbaar voor iedereen. (In de oude opzet zijn ze ook zichtbaar maar kan niet direct worden afgeleid waar de merged code vandaan kwam. In plaats van dat er een "ghost repo" van ander team in de commit message staat.)

Nadelen

- De initiële overzet naar deze nieuwe structuur is even wennen en zal één keer even wat tijd kosten.
    - Van ieder team moet de repository beheerder een fork maken van de *"main repository"*. (2 minuten werk)
    - Ieder team moet nieuwe code, die nog niet verwerkt was in het samengevoegde resultaat, hier naar verhuizen. Dit zou geen moeite moeten zijn: copy paste de eigen team package (control, analyse, define, validate, graphics) naar de nieuwe fork van de repository en testen van het resultaat. (1-2 uur werk)
- Er is een korte relatie tussen de *"main repository"* en de forks. Fouten zouden ietsje sneller de repository kunnen komen, maar dit blijft een onderdeel waar Team 1 rekening mee moet houden. Eerst moeten teams de juiste versies via de de team repositories beschikbaar stellen, nu moet dat via pull requests die niet via repositories, maar via commits werken.
- De forks staan op naam van een persoon, niet de "organisatie." Dit kan een probleem zijn, maar zou het niet verwachten. Omdat alle code uiteindelijk toch in de *"main repository"* terecht komt maken de forks na de ontwikkeling van de applicatie niks meer uit.

Conclusie

Minder nadelen dan voordelen, maar ik kan niet garanderen dat ik iets over het hoofd gezien heb. Het klopt dat er in het begin wat onduidelijkheden waren over de mogelijkheden op Github, maar deze zouden nu geëlimineerd moeten zijn.

Er kunnen geen meerdere forks op één account van dezelfde repository waardoor deze op accounts van projectleden komen te staan. Een pull request is geen magische “click to fix all problems” functionaliteit. Er zal dus tijdens de rest van het project goede controle moeten worden uitgevoerd op de pull requests en de inhoud van de Git repositories.

Wij willen hier ons nog voor inzetten vooral uit belang voor de afloop van het project, het voorkomen van problemen in de toekomst. Beter voorkomen dan genezen. Echter als deze nieuwe methode als te veel werk wordt gezien leggen wij ons neer bij die beslissing.

Auteur notities:

Overal is de tekst “main repository” tussen aanhalingstekens geplaatst omdat Git een distributed SCM is.

Workflow versie 2

Als deze methode wordt aangenomen dan verwachten wij dat deze onderstaande workflow wordt nageleefd door alle teams.

Forken *"main repository"*

1. Open de Github repository in de browser.
    1. https://Github.com/HUSACCT/HUSACCT
2. Klik op "Fork".
3. Github gaat nu een fork op jouw account plaatsen.
    1. 1-2 min later stap 4.
4. Klaar, eigen fork van de nieuwe *"main repository"*.
5. Voeg eigen teamleden toe als collaborators.

Overzet nieuwe methode

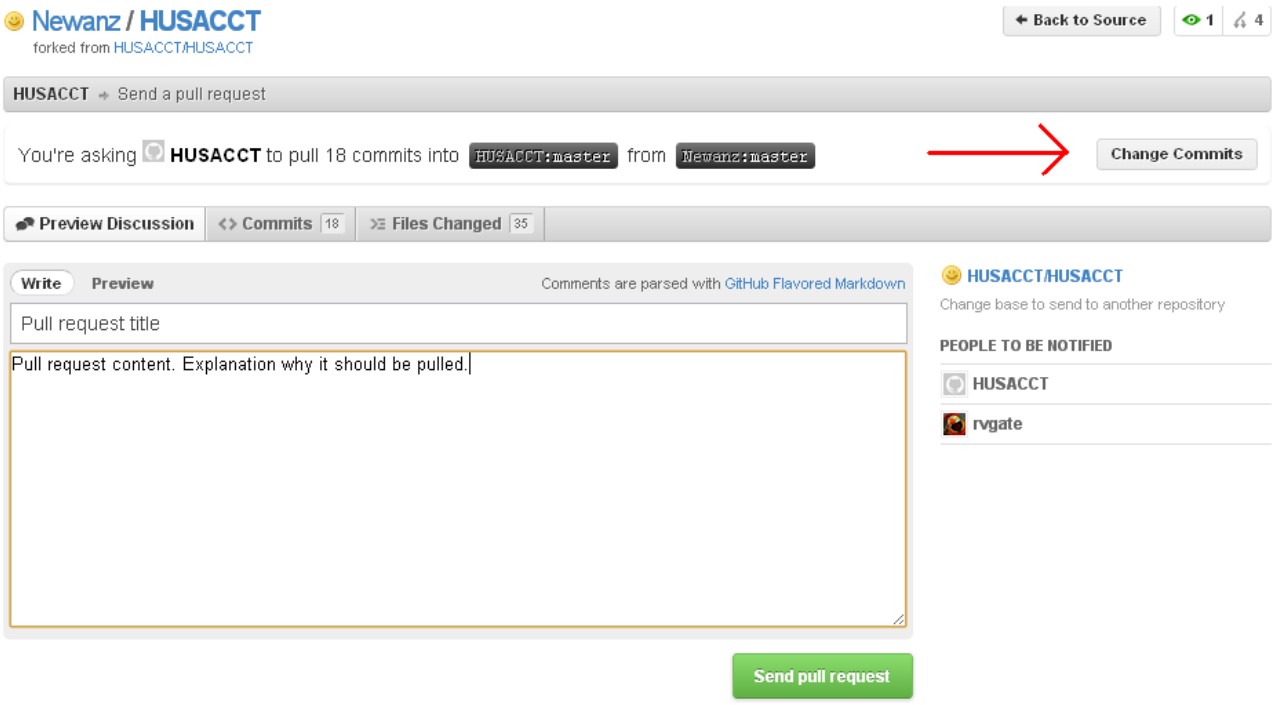
Deze situatie gaat er van uit dat er nog gecommit is op de huidige situatie en dus op een team fork zoals aangegeven in het figuur bij "Huidige situatie". De gewijzigde code zal dus moeten overgezet worden naar de fork. Dit hoeft maar door één teamlid, de git repository owner, uitgevoerd te worden.

1. Voer eerst Forking *"main repository"* uit.
2. Clone de fork naar je machine.
    1. Git haalt de fork repository naar jouw computer.
3. Zorg dat je oude fork, gebaseerd op: https://Github.com/HUSACCT-Teams/HUSACCT_Team_#, lokaal up-to-date is met alle aangebrachte wijzigingen.
4. In explorer, kopieer de package van jouw team (control, analyse, define, validate of graphics) naar de locatie van de nieuwe fork. Natuurlijk kopieer je je team package over de gemergde code in de main repo.
5. Importeer de nieuwe fork in Eclipse.
6. Verhelp build/compile problemen die Eclipse genereert.
    1. Voeg niet bestanden toe zoals:
        1. Code die niet overeenkomt met de afgesproken project structuur. Zoals Team 3's Main package. Dit is een prototype/test implementatie voor dit team en is niet relevant om opgenomen te worden in de Git repository.
        2. Test code. (sorry team 3 weer)
            1. De benchmark_applicatie directory is alleen relevant in de ontwikkeling van de applicatie.
    2. Een oplossing voor dit soort dingen is door directories en files te linken in Eclipse. Sleep een directory van explorer naar je Eclipse project en kies "Link to files and folders." Nu worden de bestanden niet meer opgenomen in de Git repo.
7. Commit de wijzigingen en push deze naar de fork.
8. Plaats een pull request op de *"main repository"* door op de eigen fork op de knop "Pull request" te klikken.
    1. Zie workflow nieuwe methode (volgende kop) punt 3 hoe je een pull request instuurt.
9. Wacht tot deze wordt uitgevoerd.

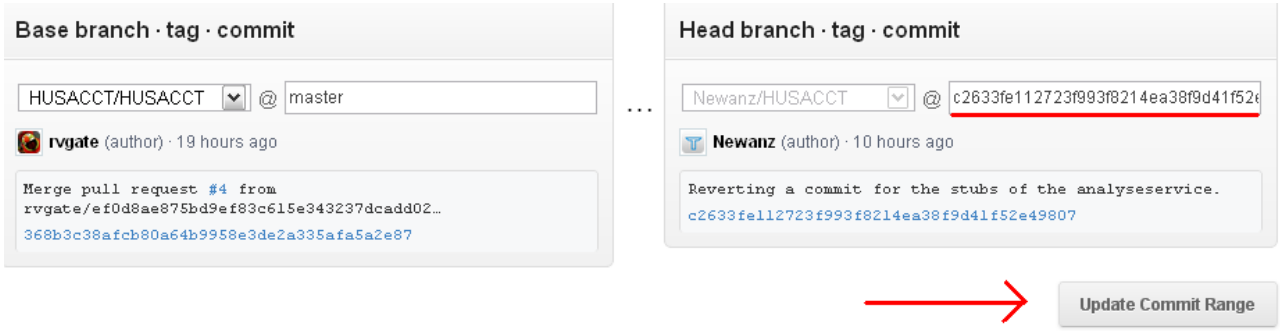
Workflow nieuwe methode

Veel of de methode is hetzelfde als eerst, alleen de structuur van de Git repository is anders.

1. Voer eerst “Forking 'main repo'” en “Overzet nieuwe methode” uit.
2. Git
    1. Zorg dat je in de nieuwe fork van jouw team werkt.
    2. Wijzig files.
    3. Commit files.
    4. Pull commits (van je teamgenoten) van de fork.
        1. Los merge conflicten op.
    5. Push commits naar de fork.
3. Stuur een Github pull request in. Dit doe je zodra je verwacht een stabiele versie te hebben.
    1. Stuur een Pull request naar de *“main repository”* op https://Github.com/HUSACCT/HUSACCT.
    2. Druk op Pull request.
    3. Een pull request kan op een aantal manieren worden ingediend.
        1. Geef een beschrijving van de pull request op. Wat er veranderd is sinds de laatste keer (of sinds de merge).
        2. Geef aan welke commit (via de hash) gepulled moet worden in de pull request. Dit kunnen ook branches zijn (master branch is standaard).
            1. Het kan dus voorkomen dat je niet de allerlaatste versie zoals beschikbaar is op de fork wil dat gepulled wordt, zeker als alle teamgenoten nog verder werken op de master branch. Geef dan een commit hash op i.p.v. de branch “master” bij het insturen van het pull request.
                1. Zie plaatje, dit moet grotendeels overeenkomen met je eigen scherm als je op “pull request” drukt.



2. Als je aan wil passen wat er gepulled moet worden druk je op “Change commits” rechtsbovenin. Het scherm in onderstaand plaatje verschijnt. Aan de rechterkant geef je op wat je wilt dat gepulled wordt op de main. Standaard staat hier “master”. Je kan een specifieke commit opgeven door de hash in dit veld te kopieren.



1. De commit hash kan je vinden in
    1. Git (git log);
    2. op Github op de “Code” pagina en vervolgens het tabje “Commits”.
2. Daarna druk je op “Update commit range” en verdwijnt de commit range selection tool. Je kan zien wat er wordt meegestuurd in het pull request onder het tabje “commits” en “files changed” op de pull request page.

3. Na het correct hebben ingediend van een pull request kan over deze pull request gediscussieerd worden op Github zelf op de pull request pagina. Als deze niet akkoord is gevonden dan wordt deze gesloten zonder gepulled te worden.

4. Team 1 merge
Na het binnen krijgen van een pull request gaat Team 1 ervoor zorgen dat het gemerged wordt in de “*main repository*”. Dat kan niet zomaar.
    1. Bekijk en beoordeel pull request.
    2. Voer een test uit van de pull request.
        1. Merge de fork op de commit die ingediend is.
        2. Laad het project in in Eclipse en controleer de build status van het project.
        3. Zijn er test beschikbaar -> run alle tests.
        4. Is de build status failing of failen de tests?
            1. Informeer het team dat de pull request heeft ingediend om de fouten te verhelpen.
            2. Sluit het pull request.
        5. Is de build status passing en passed de tests?
            1. Accepteer de pull request op Github.
            2. De teams kunnen deze wijziging zien op Github, via het dashboard, maar omdat niemand tot bijna niemand dit waarschijnlijk gebruikt moet er een mail verstuurd worden naar de team Git repository beheerders **en** architecten.
                1. Team 1 – Control
                    1. Architect: René van Aerle (rene.vanaerle@student.hu.nl)
                    2. Git: René van Aerle (rene.vanaerle@student.hu.nl)
                2. Team 2 – Validate
                    1. Architect: Maarten van Diemen (maarten.vandiemen@student.hu.nl)
                    2. Git: Jorik Kraaikamp (jorik.kraaikamp@student.hu.nl)
                3. Team 3 – Analyse Java
                    1. Architect: Rens Groenveld (rens.groenveld@student.hu.nl)
                    2. Git: Tim Muller (tim.t.muller@student.hu.nl)
                4. Team 4 – Define
                    1. Architect: Bob Sanders (bob.sanders@student.hu.nl)
                    2. Git: Henk ter Harmsel (henk.terharmsel@student.hu.nl)

5. Team 5 – Architecture Graphics
    1. Architect: Guido van Tricht (guido.vantricht@student.hu.nl)
    2. Git: Tom de Bruijn (tom.debruijn@student.hu.nl)
6. Team 6 – Analyse .NET
    1. Architect: Mittchel van Vliet (mittchel.vanvliet@student.hu.nl)
    2. Git: Martin van Haeften (martin.vanhaeften@student.hu.nl)

5. Andere teams (1-6)
   Pull wijzigingen van de "*main repository*" naar de team fork. Als dit niet gebeurt krijgt de team fork nooit de wijzigingen van de merge op de "*main repository*" binnen. Voer dit dus uit: het liefst voor het insturen van een pull request en na de gekozen datum voor de merge.
    1. Pull vanaf de *"main repository"*. Er is een pull request/merge uitgevoerd op de "*main repository*", er zijn dus nieuwe gegevens beschikbaar voor ieder team om te gebruiken. (service updates etc.)
        1. Open de team fork "team#/HUSACCT" in Git (Git bash/een Git gui). (lokaal op het systeem)
            1. Of de merge uitgevoert wordt op de master branch of op een development branch is niet belangrijk, zolang het eindresultaat op de master branch beschikbaar komt.
        2. Merge de main repository met de eigen fork. We gaan er hier vanuit dat de main branch op de "*main repository*" de laatste stabiele gemergde versie is, zoals gedefinieerd in deze workflow.
            1. Git bash
                1. Voer het volgende commando uit: git pull git://github.com/HUSACCT/HUSACCT.git
            2. Een Git GUI
                1. Pull vanaf de volgende url "git://github.com/HUSACCT/HUSACCT.git"
        3. Er zal geprobeerd worden een nieuwe merge commit aan te maken.
            1. Als dit niet mogelijk is, door merge conflicts, komt er een melding hiervan. Eerst zullen de conflicts opgelost moeten worden voordat de merge afgerond kan worden. Hoe conflicten opgelost moeten worden kan niet op een "file by file basis" worden beschreven.
        4. Nu is de team fork geüpdate met de "*main repository*" door een pull vanaf de "*main repository*" binnen te halen en deze mergen met de fork. Dit resultaat is dus alleen op de fork beschikbaar en niet op de "*main repository*".
        5. Push de wijzigingen (de merge vanaf de "*main repository*") naar de team fork zodat de rest van het team er ook bij kan.
        6. Als er wijzigingen gedeeld moeten worden met de "*main repository*", gaan we naar stap 3.
6. Begin weer bij stap 2.

.gitignore

Dit is de leading .gitignore file. Gelieve hier geen eigen entries in maken, zie bovenstaande (gehele) workflow voor exceptions in de project path.

```
bin
.classpath
.project
.settings
*.log
```