



Opleiding Applicatie Ontwikkelaar

Leerlijn Documentatie Versiebeheer & Scrum

Samen projecten doen

Domein D t/m G Level 2

Auteur: Erik Mast, Aminah Balfaqih

Datum: 5-10-2018

Inhoudsopgave

Overzicht	4
Voorkennis.....	4
Combineren met.....	4
Materialen	4
Bronnen	4
Instructies	4
Einddoelen.....	4
Beoordeling	4
Documentatie.....	5
Functioneel Ontwerp.....	5
Technisch Ontwerp.....	7
Scrum.....	10
Inleiding	10
Aan de slag	10
Het Functioneel Ontwerp schrijven	10
User-Stories	11
Groeperen van user-stories.....	11
Programmeren!	12
Versiebeheer	13
Doelen	13
Beoordeling	13
Uitleg	13
GIT in een team (+/- 16 uur).....	14
Werken in GitHub.....	15
Oefening 1: Samenwerken kun je leren	17
Problemen in Git	18
Git handleiding voor ongelukken	19
Oefening 2: Omgaan met problemen	20
Eindopdracht (in tweetallen).....	21
De opdracht.....	21
Inleveren.....	22



Bronnen	23
---------------	----

Overzicht

Level: Domein D t/m G Level 2
Duur: Onbepaald
Methode: Weekplanning

Voorkennis

- DG1 – Projecten doen
- C1, A1, B1, C1

Combineren met

Kennis uit minimaal:

- A2, B2, C2,

Materialen

- Je laptop met;
- Programmeeromgeving voor je project
- Geïnstalleerde GIT omgeving
- Een GitHub account
- Teamlid die op hetzelfde punt in zijn/haar studie is.

Bronnen

- Zie bijlage Bronnen

Instructies

- Wat is een FO (Functioneel Ontwerp) en een TO (Technisch Ontwerp)
- Zelf een FO en een TO schrijven
- Werken met versiebeheer in een klein team
 - Hoe werk je samen in Git
 - Spelregels afspreken
- Scrum voor een klein team

Eindoelen

- Je kunt zelf een FO en TO schrijven
- Versiebeheer:
 - Je kunt met iemand samenwerken door te gebruik te maken van Git.
- Scrum: Je kunt een programma beschrijven in user-stories en die groeperen zodat je een samenhangende lijst van user-stories krijgt die je samen kunt bouwen in sprints

Beoordeling

Deze module wordt beoordeeld aan de hand van een programmeeropdracht, waarbij je kunt laten zien dat je de kennis kunt gebruiken en beheerst, zowel op het gebied van Versiebeheer als Scrum

Na voltooiing van de eindopdracht worden alle domeinen (D t/m G) in een keer afgetekend. Voor zover van toepassing wordt een cijfer toegekend voor alle domeinen.

Documentatie

Documentatie is een onderschat onderdeel van je werk als programmeur. Je zult heel vaak in de situatie komen dat er geen documentatie is, of verouderde documentatie. Maar het hoort erbij. ("Ik vind het niet leuk" is geen excuus, hooguit een reden om direct ermee te beginnen!)

Vaak wordt dit naar achteren geschoven, omdat het levende documenten zijn, maar het is verstandiger om er zo snel mogelijk aan te beginnen. Beter een document dat niet helemaal af is, dan een ontbrekend document. En documenten zullen de overdracht van een project (en ook dat gebeurt in het echt!) eenvoudiger maken, en ze zullen de nieuwe programmeur sneller op weg helpen.

Elk project document voldoet aan een paar eisen:

- Het heeft een voorblad en bevat
 - De naam van het project
 - Titel van het document (bv Functioneel Ontwerp)
 - Namen van de schrijvers
 - Versienummer van het document
- Een revisietabel (wie heeft op welke datum wat aangepast?)
- Een inhoudsopgave (dit kun je zelfs automatisch laten genereren in Word met hoofdstukken en paragrafen. Tip: maak een document waarin dit werkt, en begin daarna pas te typen)
- Heeft paginanummers.

Functioneel Ontwerp

Voor je een programma kan schrijven, moet je een idee hebben waartoe het moet dienen en hoe het moet functioneren. Daarom moet er eerst een behoefteanalyse gedaan worden. De behoefteanalyse inventariseert alle functies die men in het programma wil hebben en wordt gedaan door het afnemen van interviews met de betrokken gebruikers.

Bij het ontwikkelen van software is de functioneel ontwerp zowel voor de ontwikkelaars als voor de andere betrokken partijen een belangrijk referentie document.

Doel van het document

In het functioneel ontwerp moet duidelijk worden wat de wensen en eisen aan het project en/of eindproduct(en) zijn. Het moet een gedetailleerd beeld geven van wat de opdrachtgever wil/gaat krijgen. Je kunt tevens oplossingen aangeven of aanvullende wensen en/of eisen adviseren. Geef dit wel duidelijk aan en licht het toe bij de presentatie van het Functioneel Ontwerp bij de opdrachtgever.

Inhoud van het document

Opdrachtgever

Beschrijf de opdrachtgever aan de hand van organisatie (afdeling), werkzaamheden, producten en/of diensten. Kijk daarbij voor al naar die zaken die van belang zijn voor het project, maar geef ook voldoende algemene informatie zodat de plaats en het belang van het project binnen en voor de opdrachtgever duidelijk wordt.

Het kopje opdrachtgever in de definitiestudie beschrijft het bedrijf algemeen en de beschrijving. Binnen het functioneel ontwerp kan een specifieke afdeling of een medewerker zijn binnen het bedrijf. Je moet weten voor wie je het werk gaat doen.

Algemene eisen

Wat zijn de eisen en wensen die je van de opdrachtgever krijgt en wat is zijn doelstelling voor het project, deze kan namelijk anders zijn dan die van de eindgebruikers. Denk hierbij aan efficiëntie, beveiliging, kwaliteitsbewaking, flexibiliteit, beheersbaarheid, prestaties, opmaak en de gebruikersinterface.

Eindgebruiker(s)

De eindgebruikers zijn de mensen die uiteindelijk gebruik gaan maken van het eindproduct. Verschillende eindgebruikers hebben vaak verschillende eisen en wensen, maar vooral ook verschillende verwachtingen van het eindproduct. Het is belangrijk daarvan een compleet beeld te krijgen zodat je niet een project gaat uitvoeren waarvan niemand het eindproduct wil gebruiken. Vergeet geen gebruikers; een klant van een webwinkel is ook een gebruiker van dat informatiesysteem, de opdrachtgever is vaak ook gebruiker bijvoorbeeld als hij managementinformatie uit het systeem wil halen.

Wat zijn de verwachtingen (wensen en eisen) van de eindgebruiker(s) van het project/eindproduct

Ga per eindgebruikersgroep bekijken wat ze verwachten van het eindproduct. Denk daarbij aan functionaliteit; hoe gaan ze gebruikmaken van het eindproduct, op welke momenten, bij welke werkzaamheden. Belangrijk is dat je kijkt met de ogen van de eindgebruiker; wat zien ze van het eindproduct, bijvoorbeeld de lay-out van gebruiksinterface, de formulieren, de rapporten of de tekst van een handleiding. Eén eindgebruiker kan verschillende momenten een andere functionaliteit verwachten.

Wanneer je een nieuwe telefoon besteld bij een webwinkel verwacht je een ander scherm dan wanneer je de bestelstatus, de verzendstatus of je factuur wilt zien. Welke informatie wil de gebruiker uit het systeem halen?

Technisch Ontwerp

Een technisch ontwerp bevat de meer technische zaken. Denk aan systeem eisen en software versies.

Maar ook aan bijvoorbeeld database-diagrammen (hoe zijn de tabellen met elkaar verbonden met foreign keys) en object diagrammen (hoe erven de objecten van elkaar over) en object-interactie diagrammen (hoe werken de objecten samen in de software, dit gedeelte is meestal generiek, beschrijvingen voor verschillende programmaonderdelen die steeds hetzelfde werken, worden niet herhaald)

Doel van het document

Het maken en verantwoorden van technische beslissingen voor het project/eindproduct zodat functionele eisen en wensen gehaald gaan worden.

Inhoud van het document

Toelichting gekozen oplossing(en)

Geef hier een samenvatting van alle gekozen oplossingen.

Eindproduct(en)

Een beschrijving van alle eindproducten. Geef per eindproduct de inhoud aan, om daarna per eindproduct de systeemonderdelen te (be)noemen.

Systeemonderdelen / Objecten

Welke losse systeemonderdelen / objecten kun je onderscheiden? Bekijk welke onderdelen/objecten van het eindproduct je kunt beschrijven. Van elk onderdeel/object moet je duidelijk de functie en zijn belangrijkste eigenschappen noemen. Door een duidelijke lijst van onderdelen/objecten te maken kun je later een planning maken waarbij verschillende personen verantwoordelijk zijn voor de verschillende onderdelen.

Bijvoorbeeld:

- Uit welke hardwarecomponenten bestaat je systeem; webservers, applicatieservers, databaseservers en clients?
- Welke database wordt gebruikt?
- Welke formulieren worden er gebruikt? Welke pagina's omvatten de applicatie? Welke klassen moeten er komen? Objecten benoemen zoals variabelen, sessies, functies en methoden (belangrijk wanneer de info door verschillende programmeurs gebruikt dient te worden)

Kwaliteitseisen

Welke (kwaliteits)eisen stel je aan elk systeemonderdeel? Probeer zoveel mogelijk te denken aan specifieke nummers, bijvoorbeeld hoeveel, hoe groot, hoe snel, hoe lang, hoe duur...? Aan de hand van deze gegevens kun je ook later je systeemtest gaan inrichten.

Bijvoorbeeld:

- Hoeveel clients hebben we nodig?
- De locatie van de database, de pagina's etc. Noteer ook eventuele hostinggegevens en/of wachtwoorden.
- Hoeveel gebruikers gaan er gebruik maken van de applicatie?
- Welke afspraken gelden er? Denk hierbij aan afspraken over includes, functies, opmaak, variabelen, naamgeving en instellingen.
- Hoe wordt er omgegaan dataopslag? Denk hierbij aan het opslaan van afbeeldingen.

Afhankelijkheden

Welke relatie bestaat er tussen de verschillende systeemonderdelen? Hoe is een onderdeel afhankelijk van andere onderdelen? Door dit goed te beschrijven wat je vooraf moet vastleggen om de onderdelen apart te kunnen maken. Daarnaast geeft dit veel informatie voor de systeemtest. Behalve dat het onderdeel goed moet werken volgens de specificatie moeten de onderdelen ook goed samenwerken. Hierop kun je tijdens de systeemtest ook testen.

Bijvoorbeeld:

- Zijn er afhankelijkheden vanuit een andere applicatie? (data-upload vanuit een ander pakket?)
- Zijn er afhankelijkheden naar een andere applicatie? (data-output naar een ander pakket?)
- Bestaat er een relatie met een ander pakket?
- Vindt er een conversie plaats? Moet er rekening gehouden worden met oude data?

Structuren

Welke structuren kun je onderscheiden? Door de structuren uit te werken in schema's kan er veel duidelijk worden over de interne werking van het systeem. Dit levert veel informatie op over de kwaliteitseisen die je moet stellen aan de systeemonderdelen. Bijvoorbeeld een netwerkstructuur geeft veel informatie voor het bepalen van de snelheid van switches en bekabeling. Een stromingsdiagram voor gegevens geeft veel informatie hoe deze door een applicatie lopen. Daarmee kan bepaald worden welke modules wat welke manier specifieke gegevens moet verwerken.

Bijvoorbeeld:

- Datamodel met uitwerkingen per veld
- Linkenplan, boomstructuur
- Klassendiagram
- Usecasediagram (gebruikersscenarios)

Scrum

Inleiding

In principe is Scrum met zijn tweeën hetzelfde als in je eentje. Er is één verschil: je moet met iemand samenwerken. Dat betekent dat je samen de eisen opstelt (het FO schrijven) samen bepaalt hoe het technisch gaat werken (TO opstellen) en ook samen de afzonderlijke taken bepaalt.

Lees daarom eerst nog eens de module DG1 door, en dan met name het Scrum gedeelte.

Aan de slag

De eerste stap die gedaan moet worden is dat jullie in kaart brengen welke functionaliteit je gaat bouwen. Maak dus samen een lijstje van wat het programma moet gaan doen.

Bijvoorbeeld:

- Contact maken met de database
 - Klasse maken die verantwoordelijk is voor de inloggegevens op de database
 - Klasse maken die queries kan versturen naar de database en eventueel de gegevens ophalen (probeer dat zo generiek mogelijk te doen, je zult waarschijnlijk meer dan één tabel gebruiken)
 - Een klasse maken die een gegevensrij vertegenwoordigt in de database
- Gebruikerseisen vertalen
 - Een gebruiker wil kunnen inloggen
 - Een gebruiker wil zijn gegevens kunnen aanpassen
 - Een gebruiker wil zijn wachtwoord kunnen wijzigen
 - Een gebruiker wil zijn account op non-actief kunnen zetten
 - Een beheerder wil een gebruiker kunnen toevoegen
 - Een beheerder wil een overzicht hebben van alle gebruikers
 - Een beheerder wil een gebruiker kunnen aanpassen
 - Een beheerder wil een gebruiker op non-actief kunnen zetten

Het bovenstaande voorbeeld is natuurlijk verre van compleet.

Het Functioneel Ontwerp schrijven

In Scrum termen is een FO relatief simpel. Je beschrijft alleen de algemene zaken, en alles wat je kunt beschrijven als “Een gebruiker wil...” noemt men ook wel User-stories en worden in losse documenten of bijlages uitgewerkt.

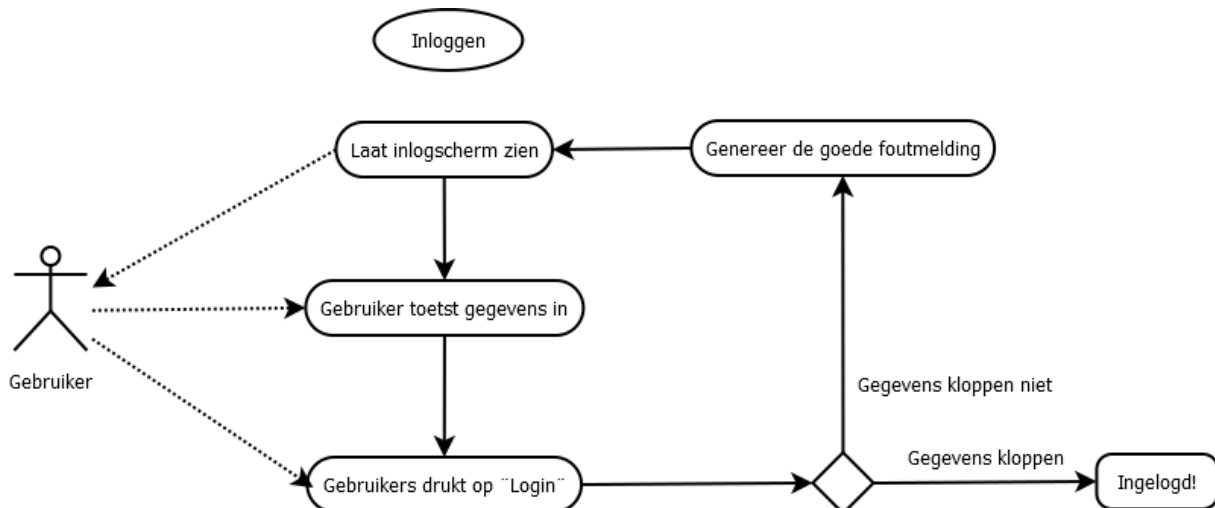
User-Stories

Een User-Story is een beschrijving van het proces. Stel we kiezen het proces “Een gebruiker wil kunnen inloggen”

Dit zou je kunnen beschrijven als:

- Een gebruiker wil kunnen inloggen
 - Gebruiker klikt op “Inloggen”
 - Gebruiker krijgt het inlogscherf te zien
 - Gebruiker vult zijn naam en wachtwoord in
 - Gebruiker klikt op “Login”
 - Het gaat goed: gebruiker is ingelogd
 - Het gaat fout: gebruiker krijgt een foutmelding en keert terug naar het inlogscherf

Je zou het ook kunnen tekenen, bijvoorbeeld zoals onderstaand:



Nu heb je al bedacht hoe het ruwweg werkt, en wanneer de gebruiker iets te zien krijgt of iets moet doen. Met de bovenstaande beschrijving of tekening kan dus (bijna) iedereen dat stukje programma al testen! Het moet alleen nog geschreven worden

Groeperen van user-stories

De volgende stap is het groeperen van user-stories. In het bovenstaande voorbeeld groepeer je user-stories in volgorde van hoe belangrijk ze zijn.

Dus voor de eerste ronde bouw je waarschijnlijk een stukje programma zodat je alles in de database kunt opslaan en gegevens kunt ophalen, en als onderdeel daarvan maak je een “gebruiker” tabel aan. Met een bijbehorend “gebruiker” object in de code, waarin je de gegevens kunt bewaren die je uit de database haalt. Het eerste stukje is altijd lastig om in Scrum te vatten, maar door goed samen te werken (in Git) zou dat moeten lukken.

In de tweede ronde zou je bijvoorbeeld alle gebruikers functies kunnen maken, en die kun je dan in je team verdelen



De derde ronde zou zijn voor alle beheerders functies. **Want een beheerder is ook een speciale gebruiker!**

De drie rondes hierboven worden ook wel “Sprints” genoemd. Een sprint is een periode waarin geprogrammeerd en getest wordt. Kenmerk van begin en eind van de Sprint is, dat aan het begin de software werkt, en aan het eind van de sprint werkt de software ook weer helemaal inclusief de nieuwe dingen die zijn toegevoegd. In die Sprint zitten een beperkt aantal taken, die meestal functionele samenhang hebben (dus bijvoorbeeld alle gebruikers functies) of een andere samenhang (bijvoorbeeld omdat de minister een nieuwe wet heeft aangenomen die ingaat op een bepaalde datum). Binnen de Sprint wordt er ook voor gezorgd dat er genoeg taken zijn om de Sprint vol te maken. Daarom worden alle taken ook ingeschat (van makkelijk tot moeilijk)

Programmeren!

Als je dit allemaal geregeld hebt, zou je aan de slag kunnen gaan met programmeren.

Versiebeheer

In deze module leer je hoe je kunt samenwerken door bij het programmeren gebruik te maken van versiebeheer. We doen dit a.d.h.v. Git en GitHub

Doelen

Na deze module kun je in een project waarin je samenwerkt Git gebruiken.



Beoordeling

Deze opdracht wordt beoordeeld aan de hand van de commits die je voor dit project doet in Git. Doorslaggevend zijn

- Zinnvolle commentaren
- Evenwichtige samenwerking (d.w.z ieder teamlid draagt evenveel bij aan het project)
- Het is niet toegestaan om twee projecten parallel te ontwikkelen zonder gedeelde code, er moet echt in **één** project gewerkt worden

Uitleg

In deze module wordt zoveel mogelijk gebruik gemaakt van icoontjes om iets duidelijk te maken:

	Dit tekstgedeelte is van toepassing als je in de command prompt met Git werkt
	Als je met TortoiseGit werkt moet je deze opdrachten uitvoeren.

GIT in een team (+/- 16 uur)

Laten we simpel beginnen: samenwerken in een team terwijl je gebruik maakt van GIT is bijna hetzelfde als in je eentje GIT gebruiken op twee computers. Er is één groot verschil:



Er wordt tegelijkertijd aan de code gewerkt door meer personen.

Dit betekent dat de code die op jouw computer staat niet de laatste code hoeft te zijn, en niet correct hoeft te zijn.

Daarom heb je spelregels nodig:

Start de dag

Als je begint met programmeren aan je project moet je eerst een **pull** doen:

	<p>Ga in de command prompt naar de map waar je project staat, en voer deze opdracht uit:</p> <pre>git pull origin master</pre>
	<p>Klik rechts op de map waar je project staat en kies TortoiseGit-> Pull</p>

Dit zorgt ervoor dat alle code (en andere informatie) in je werkmap terecht komt, en je dus fris aan de gang kunt. Je kunt code bestanden toevoegen met **add** of aanpassingen doen en die in je lokale repository indienen met **commit**

Werk in samenhangende code

Als je aan het programmeren bent, zorg dat datgene wat je doet alleen te maken heeft met de taak waaraan je werkt. Dat voorkomt veel problemen met commiten etc. Dan heb je altijd een samenhangende set van wijzigingen hebt, die je opstuurt naar de repository op GitHub.

Als je taak af is, dan doe je **push**.

In de problemen

Het kan zijn dat je problemen krijgt als je een **pull** doet. Dan moet je dingen mergen. Meestal krijg je hiervan een melding als het niet automatisch kan, en dan kun je beste even ervoor gaan zitten met een glas ranja, want je wilt dit wel secuur doen. Als je fouten maakt kan dat vervelende consequenties hebben voor je project.

Aanpassingen voor een merge kun je beste meteen opsturen naar de GitHub repository, tenzij je nog bezig bent om een taak af te ronden.

Werken in GitHub

In GitHub kun je behalve code ook een sprintboard bijhouden. Taken die je doet voor je project heten in GitHub “Issues”. Dus als er in deze module gesproken wordt over taken en je werkt in GitHub dan bedoelen we issues.

Sprintboard maken

De taken hebben voor je project niet zoveel samenhang, dus maak je ook in GitHub een project aan met dezelfde naam als je repository waarin je gaat werken. In zo’n project maak je (voor nu) drie kolommen aan, “Te doen”, “Bezig” en “Gedaan”. Als je dat gedaan hebt kun je ook nog automatisering aanleggen.

Kolom Te Doen

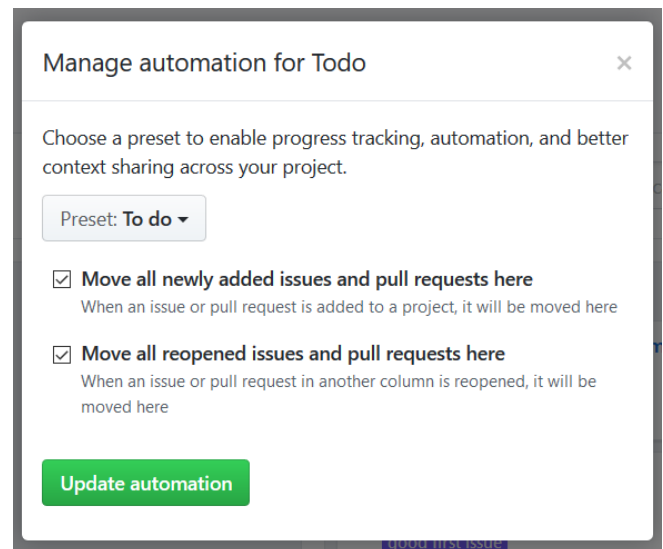
Je klikt op de ... knop van de eerste kolom en kiest “Manage Automation”. Je krijgt een scherm als hiernaast en kiest vanzelfsprekend de preset: **To Do**.

Klik beide vinkjes aan en daarna op Update.

Dit zorgt ervoor dat als er nieuwe taken gemaakt worden, die automatisch in de kolom “Te doen” worden toegevoegd.

Kolom Bezig

Hetzelfde doe je voor de kolom “Bezig” en je kiest de Preset “In Progress”. Hier hoeft je geen vinkjes aan te vinken.

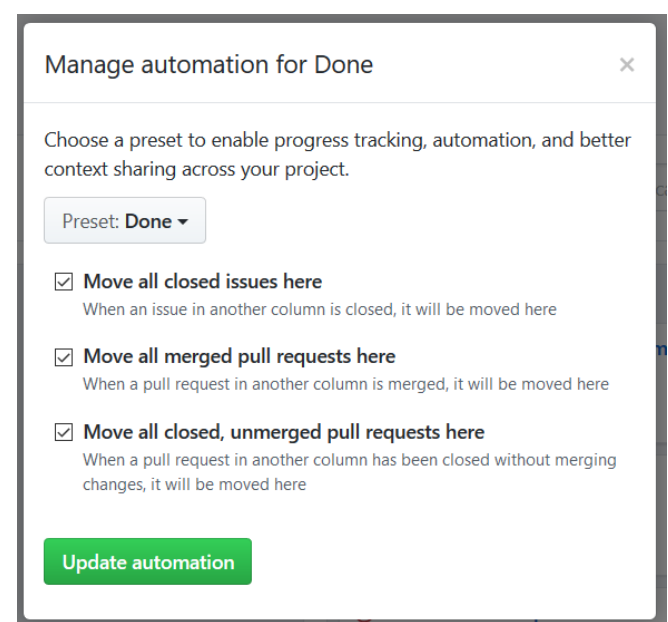


Kolom Gedaan

Voor de kolom “Gedaan” kies je de preset “Done” en zet alle vinkjes aan.

Veel ingewikkelder

Je begrijpt uit het bovenstaande dat je het veel ingewikkelder zou kunnen configureren, maar dat is hier niet nodig.





Samenwerken in GitHub

Als je wilt samenwerken in GitHub, zul je je teamgenoot aan je project moeten toevoegen. Die heten in GitHub “Collaborators”. Daar kun je je teamgenoot opzoeken.

Hints en tips

Als je bezig bent zul je er achter komen dat je steeds als je een **commit** en je hebt een taak afgerond, dat je je taken met de hand moet afronden (op “Done” zetten).

Daar is voor GitHub een truc (in combinatie met de configuratie die je hierboven hebt gedaan).

In je commentaar voeg je closed #4 toe. Dat zal er voorzorgen dat taak #4 afgesloten wordt. Als het goed is schuift de taak dan automatisch door naar de kolom “Gedaan”

Zie daarvoor ook: <https://help.github.com/articles/closing-issues-using-keywords/>

Voor het starten van een taak is helaas niet zo’n handige truc, maar een taak verschuiven van “Te doen” naar “Bezig” is ook niet zo ingewikkeld.

Oefening 1: Samenwerken kun je leren

In de volgende oefeningen gaan we een kleine applicatie ontwikkelen om te oefenen met Git en de vaardigheden uit Module G1 en G2.

De applicatie

De applicatie kan worden geschreven in Java of PHP in combinatie met MySQL.

We willen graag een wedstrijdendatabase hebben, waarin we clubs kunnen invoeren en aanpassen. Ook willen we de wedstrijden kunnen invoeren en aanpassen, ook de score moet vastgelegd kunnen worden. Wedstrijden in de toekomst moeten ook kunnen worden geregistreerd.

Het verwijderen van clubs verdient speciale aandacht! Je wordt gevraagd waarom dit zo is, en wat jouw oplossing is en waarom.

GitHub inrichten

1. Eén van de teamgenoten moet een nieuwe repository aanmaken.
2. Voeg de andere teamgenoot toe als Collaborator
3. Maak het project aan en voeg de drie kolommen toe zoals hierboven al genoemd is

De opdracht



1. Begin met een database ontwerp. Maak daarvoor twee taken aan, voor elke tabel één. Denk aan de volgorde, want “wedstrijden” kunnen pas gemaakt worden als de “clubs” klaar zijn. De score wordt nu eerst opgeslagen in het formaat zoals 1-2 in één veld.
2. Deel de rest van de opdracht op in verschillende taken, voor zover mogelijk die onafhankelijk zijn van elkaar. Let ook op de volgorde waarin ze uitgevoerd moeten worden. **Er worden minimaal 10 taken verwacht.** Denk aan afzonderlijke taken voor toevoegen, overzicht maken, aanpassen en verwijderen van regels in de database.
3. Zet deze taken in je sprintboard, of in de takenlijst van GitHub, geef ook hier in al een werkvolgorde aan.
4. Ga samen programmeren en doe per taak een **commit**, en een **push**.
5. Als een taak afgerond is, doe dan een nieuwe net zo lang tot er 8 taken af zijn (de rest bewaar je voor de volgende oefening).

Problemen in Git

Als je wat meer met Git werkt kan het zijn dat je tegen problemen aanloopt. Daar is geen standaard oplossing voor omdat die problemen vaak verschillende oorzaken kunnen hebben. Hieronder worden een paar standaard dingen besproken.

Ik heb per ongeluk iets gecommited wat niet de bedoeling was

Je hebt als laatste iets gecommited wat niet de bedoeling was. Als je niet **push** hebt gedaan, kun je:

	<p>Ga in de command prompt naar de map waar je project staat, en voer deze opdracht uit:</p> <pre>git revert HEAD</pre>
	<p>Klik rechts op het bestand waar je per ongeluk commit op gedaan hebt en kies TortoiseGit-> Show Log</p> <p>Je krijgt een scherm met commits op dat bestand te zien en je klikt rechts op de bovenste commit (dat is de laatste, controleer dat wel) en klik op "Revert change by this commit"</p>

Deze actie voegt een nieuwe **commit** toe, waarin je laatste wijzigingen terug gedraaid zijn.

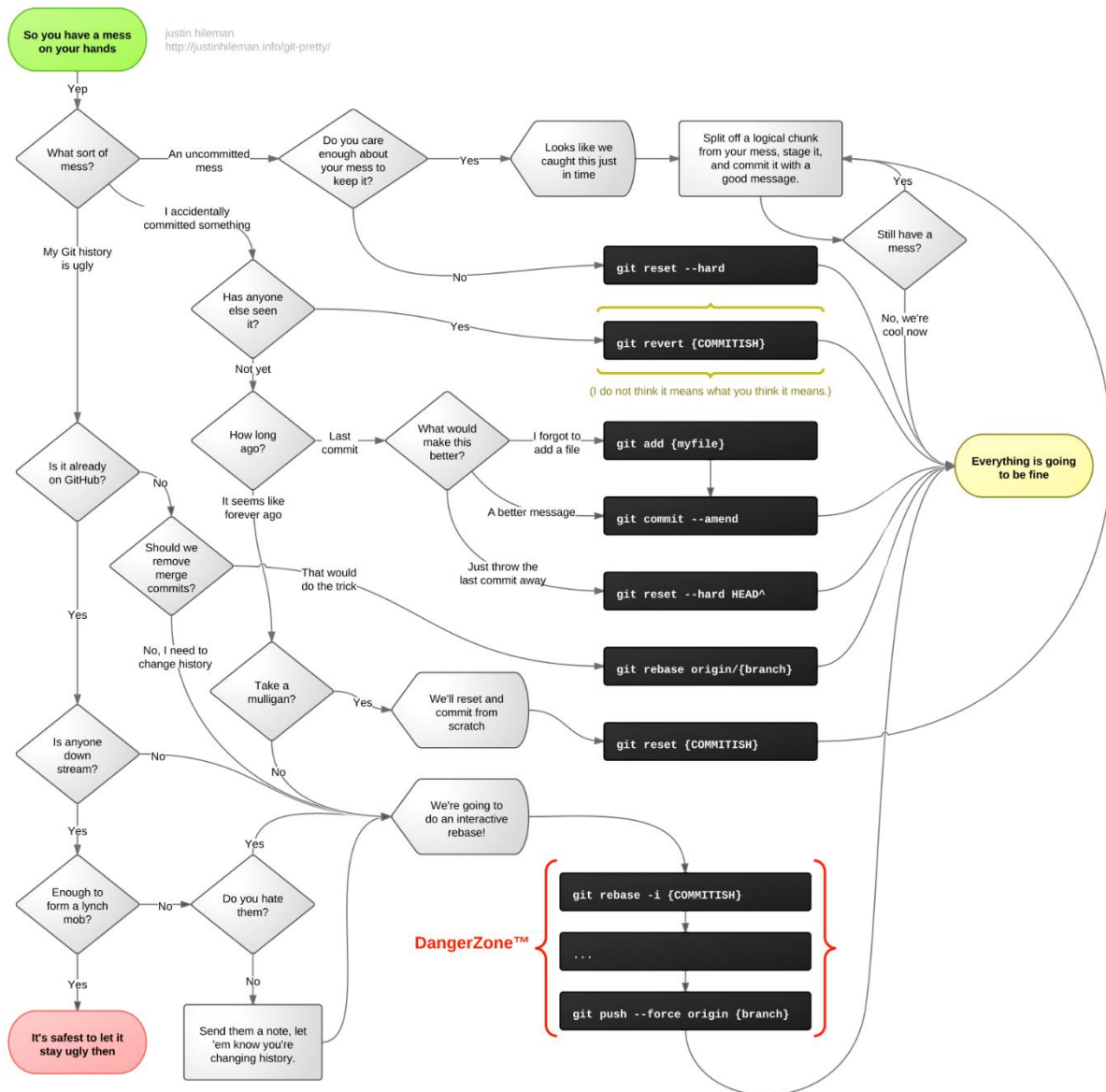
Ik heb ook nog mijn wijzigingen gepushed...

Dat kunnen we ook oplossen:

	<p>Ga in de command prompt naar de map waar je project staat, en voer deze opdracht uit:</p> <pre>git revert HEAD git push</pre>
	<p>Klik rechts op het bestand waar je per ongeluk commit op gedaan hebt en kies TortoiseGit-> Show Log</p> <p>Je krijgt een scherm met commits op dat bestand te zien en je klikt rechts op de bovenste commit (dat is de laatste, controleer dat wel) en klik op "Revert change by this commit"</p> <p>Er wordt dan een nieuwe versie gemaakt: dit wordt weer de bovenste. Klik daar op rechts en Kies dan "Commit".</p> <p>Daarna kun je rechts klikken op de map waar het project in staat: en Kies dan TortoiseGit->Push</p>

En dan zou het weer in orde moeten zijn.

Git handleiding voor ongelukken



Bron: <http://justinhileman.info/article/git-pretty/>



Oefening 2: Omgaan met problemen

Er liggen van de vorige oefening nog een paar taken. Ga met deze taken oefenen wat er gebeurt als je tegelijkertijd in één bestand werkt. Ga hier vooral veel mee oefenen. Maar vergeet niet om de taken af te maken.

Eindopdracht (in tweetallen)

Om dit level te kunnen halen gaan we een stukje estafette programmeren.

De bedoeling is dat jullie na elk stukje programmeren jouw code doorgeeft aan jouw collega student. Deze student gaat dan de code dan verder afmaken. Hierbij is het verboden om de code van de vorige student aan te passen.

Natuurlijk moet dit allemaal gepland worden zoals je geleerd hebt bij level 1. Voor eventuele informatie wordt er daarom ook terugverwezen naar de informatie in de DG1 module. Het project dient gemaakt te worden in Java.

Even de regels van dit project op een rijtje:

1. Je programmeert om de beurt in een klasse. Je mag in één beurt maar één methode maken.
2. In elke beurt beschrijf je een nieuwe functionaliteit in een taak, deze beschrijving voeg je ook toe aan het FO. Als voor die nieuwe functionaliteit nieuwe klassen noodzakelijk zijn, maak je die aan, maar je vult verder niets in. **Let op: geen programmeerwerk in de nieuwe klasse.**
3. Je zorgt dat je je code werkend aflevert in GitHub, met eventueel een nieuwe taak voor dingen die zijn blijven liggen, of niet goed werken.
4. Elke beurt duurt een half uur.
5. De volgende beurt pak je een nieuwe taak op. **Niet in de klasse waar je teamgenoot in werkt!**
6. Afronding: Zodra je programma werkt, laat je deze zien aan de docent.
 - a. Bewijs van commits is essentieel voor deze opdracht.
Geen Git commits is geen afronding.
 - b. Het programma moet werken en eventuele resterende bugs, moeten wel in GitHub gedocumenteerd zijn.
 - c. Het FO bevat alle functionaliteit die je bedacht hebt. Het TO bevat alle technische aspecten daarvan dus ook een object model.
 - d. Het object model bevat minstens een klasse voor het scherm, en een voor de logica.

De opdracht

Het is de bedoeling dat het spel galgje gemaakt wordt.

Spelregels:

- Er wordt automatisch een woord gegenereerd. In totaal zitten er 30 woorden in het systeem.
- Een speler mag 5 keer een letter invoeren. Na 5 keer wordt het woord zichtbaar en is het spel afgelopen.
- In het scherm is zichtbaar hoeveel kansen er al gebruikt zijn en welke letters er gekozen zijn.
- Elke keer als de persoon een goede letter gekozen heeft wordt deze op de corresponderende plaats van het woord gezet.
- Als de speler het woord wil raden moet hij op een knop drukken die hem de mogelijkheid geeft om het woord te raden. Is het woord fout dan wordt dit geregistreerd als een foute letter. Is het woord goed dan krijgt de speler een felicitatie en de mogelijkheid om een nieuw spel te starten.



Inleveren

De opdracht lever je in in Magister. Zorg altijd de website ingepakt verstuurd wordt (zipfile). Als je niet weet hoe je een zipfile maakt: Klik rechts op de map waar jouw bestanden staan, en kies daarna voor “Kopiëren naar”->Gecomprimeerde (gezippte) map. Er wordt dan een zipfile gemaakt die je kunt opsturen.

Bronnen

- ✓ **Fouten oplossen met Git**
<http://justinhileman.info/article/git-pretty/>
- ✓ **Git Flow commits laten zien:**
<http://beta.gitflowchart.com/>