Kwetter: Software Release Plan

David D. Diks (328689 in klas S63)

4 april 2018

1 Documenthistorie

Deze geschiedenis houdt de veranderingen aan dit document en het proces bij.

Datum	Versie	Beschrijving van veranderingen
19 februari 2017	0.1	Eerste revisie van het document

Inhoudsopgave

2 Taken en Issues

Voor het verzorgen van een geheel ontwikkelproces is er een nood voor een vorm van geschreven communicatie tussen de leden van het ontwikkelteam. Voor dit te faciliteren wordt er gebruik gemaakt van een Issue Tracker, een tool waar taken in opgeschreven worden. Om ervoor te zorgen dat deze tool leesbaar en bruikbaar is zijn hier een paar korte richtlijnen voor.

2.1 Beschrijving

Een issue staat voor een enkele "Unit of Work"[1] en beschrijft een gewenste functionaliteit. Deze kan varieëren van kleine bugfixes tot grote epics, onderverdeeld in kleinere taken.

Issues dienen als documentatie voor het project en als discussie-platform voor functionaliteit. De bedoeling is dat de issue wordt gebruikt om te discussëren over de gewenste implementatie en veranderingen toe te brengen. Na discussie dient de issue aangepast te worden conform de nieuwe besproken specificaties.

2.1.1 Standaardformaat

Een issue volgt het volgende formaat:

- Probleem/Motivatie
- Gewenste oplossing
- Mogelijke gevolgen voor de applicatie (Veranderen van dataschema of API endpoints)

2.2 Releases

Voor ondersteuning van de agile processen worden issues georganiseerd op basis van releases, om van tevoren de release te plannen waarin functionaliteit wordt geïmpementeerd. Releases hebben een naam, tag en beschrijving om in grote lijnen aan te geven welke functionaliteit bij de release hoort.
Bugfixes worden gepubliceerd onder de eerstvolgende release terwijl gewone taken een specifieke release hebben.

[1] Lees: taak

2.2.1 Versioning

Voor het dopen van een release wordt gebruik gemaakt van Semantic Versioning(Preston-Werner, 2017) om een duidelijk schema vast te leggen. De naam bestaat dus uit cijfers in het volgende formaat: **MAJOR.MINOR.PATCH**.

MAJOR staat voor een grote release die niet "backwards compatible"[2] is en kan dus enkel werken met een release van hetzelfde **MAJOR** nummer.

MINOR is een kleinere release die zonder problemen samen kan werken met release met hetzelfde **MAJOR** nummer.

PATCH is een klein en voegt geen nieuwe functionaliteit toe aan het product maar past bugfixes toe.

2.2.2 Voorbeeld

Kwetter 2.5 Implementeert veranderingen in het API schema met nieuwe authenticatie via OAuth2.

Kwetter 3.0 Doorvoeren van een complete refactor over de front-end

2.3 Tagging

Terwijl de titel van een issue de functionaliteit hiervan probeert te beschrijven kunnen vaak onduidelijkheden overblijven omtrent de taak zelf. Om dit probleem op te lossen wordt er gebruik gemaakt van een taggingsysteem waarmee de scope van een taak in een oogopslag gezien kan worden.

2.3.1 Prioriteit

Alhoewel issues al een inherente prioriteit hebben door de release is er een nood om aan te geven hoe belangrijk een taak is. Hiervoor worden de volgende tags gebruikt.

- PRIO - High
- PRIO - Medium
- PRIO - Low

2.3.2 Type

Een issue kan verschillende doelen hebben en zullen dus een andere interactie van ontwikkelaars vereisen. Dit wordt aangegeven door middle van een type

[2]Nieuwe versie werkt niet samen met oude versie

- TYPE - Feature - Nieuwe functionaliteit
- TYPE - Enhancement - Verbeteren van bestaande functionaliteit
- TYPE - Bug - Rapporteren van een bug
- TYPE - Proposal - Voorstel voor het veranderen van de applicatie

2.3.3 Module

Om te verzorgen dat issues snel gesorteerd en gevonden kunnen worden op basis van module zullen zij hiermee getagged worden. De voorbeelden die hierbij horen kunnen vaker veranderen naarmate het product ouder wordt. Deze worden dan opnieuw gedocumenteerd.

- MOD - API: Verandering in de API
- MOD - UI: User interface change

3 Ontwikkeling

Het daadwerkelijke ontwikkelproces is het belangrijkste onderdeel van software engineering en moet goed uitgevoerd worden. Om samenwerking te bevorderen wordt er gebruik gemaakt van vele conventies en best-practices om problemen in de kiem te smoren.

Waar veel mensen aanraden om met veel branches en branch sharing te werken(Driessen, 2017) zijn hier enkele inherente problemen mee. Ten eerste zorgt het voor grote integratiebranches van features. Om features daadwerkelijk te mergen kunnen er veel problemen voorkomen vanwege de vele features. Ten tweede moeten er veel branches tussen de verschillende ontwikkelaars gedeeld worden. Dit proces kost veel onnodige moeite en staat open voor problemen.(Judin, 2017)

Vanwege deze redenen wordt er gewerkt met een cactusmodel, waarbij het werkt gebeurt in de master branch. Dit haalt meerdere onnodige branches weg. Commits worden via een Staged Merge System toegevoegd aan de master branch.

3.1 Branches en Benamingen

De master-branch wordt gezien als de enkele bron van waarheid in het project waardoor al het werk in deze branch gebeurt. Om te voorkomen dat commits direct naar master gecommit worden is deze branch "protected"[1] maar worden de commits gecureerd door het Staged Merge Systeem.

Branches worden gemaakt voor releases. Deze branches krijgen de naam "release-MAJOR.MINOR.PATCH". Hotfixes worden toegepast op deze branch en cherry-picked naar master. Er gebeurt geen development op release-branches.

3.2 Commit guidelines

Om ervoor te zorgen dat commits te begrijpen zijn moeten zij leesbaar worden gehouden, anders is het in de toekomst onmogelijk om te zien wat gebeurd is. Om deze reden zijn er regels voor commits die gevolgd moeten worden.

De eerste regel van een commit is de titel en geeft een korte beschrijving van de commit zelf. Het word als samengesteld door een beschrijving van de commit (fix, feat, docs, style), de scope (cli, api, ui) en eindigend met een korte boodschap (Add endpoint

[1] Er mag niet zomaar naar gepushed worden

for serving users). Op de tweede regel wordt de commit in-depth beschreven (Add an endpoint for serving users with new serializer). Dit wordt gevolgd door een lijst met breaking changes als deze voorkomen. Als laatste heeft een commit een referentie naar een issue in de issue tracker.

3.3 Staged Merge System

Het Staged Merge System zorgt ervoor dat iedere feature in quarantaine wordt gesteld tot deze bekeken is door andere developers. Dit wordt bewerkstelligd door middel van een feature, gestopt in een branch, die klaarstaat on gemerged te worden.

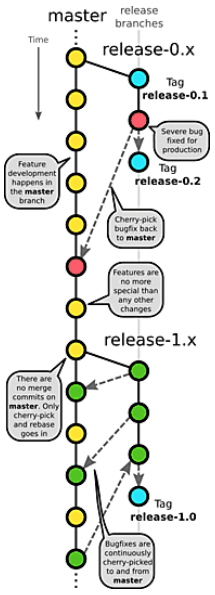
Om code te mergen in dit systeem moet er eerst een OK gegeven worden door een andere developer. Deze geeft een code-review over de commits en wacht op aanpassingen. Daarnaast wordt voor iedere commit een build getest door de CI-server. Dit wordt in gitlab verzogd door middel van Pull Requests.

4 Release

De software kan niet op een simpele manier gereleased worden aangezien deze complex is en uit meerdere onderdelen bestaat. Om te zorgen dat releases netjes gedocumenteerd en uitgerold worden is het releaseproces opgesteld. Hieronder is het proces kort beschreven.

1. Branch van master met de naam "release-MAJOR.MINOR.PATCH"
2. Voer unit tests uit op deze branch
3. Voer integratietesten uit op deze branch
4. Indien fouten opkomen zullen hotfixes worden uitgevoerd tot de testen niet meer falen
5. Tag de release met de naam "release-MAJOR.MINOR.PATCH"
6. Cherry-pick hotfixes naar de "master"branch

Dit proces maakt gebruikt van het cactusmodel voor git, waarbij het werk zelf gebeurt in de master branch en de releases apart gezet worden. Hotfixes worden teruggezet in master.



Figuur 4.1: Een voorbeeld van het cactus-model

Literatuur

Driessen, V. (2017, 19 februari). *A successful git branching model.* Verkregen van `http://nvie.com/posts/a-successful-git-branching-model/`

Judin, J. (2017, 19 februari). *A succesful git branching model considered harmful.* Verkregen van `https://barro.github.io/2016/02/a-succesful-git-branching-model-considered-harmful/`

Preston-Werner, T. (2017, 19 februari). *Semantic versioning 2.0.0: Semantic versioning.* Verkregen van `http://semver.org/`