Basis apps

Plannen en Ontwerp

hoofdstuk

4

US/Taken – Repo - Commits

Algemene informatie

Onderwerp	Taken in je userstories kun je 1:1 koppelen met commits in je repository
Leerdoel(en)	De leerling leert zijn commits van de juiste messages te voorzien zodat ze 1:1 gekoppeld zijn met een taak of soms aan een volledige userstory
Vereiste voorkennis	Agile/Scrum werken in DevOps
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☒ B1-K1-W2: Ontwerpt software ☒ B1-K1-W3: Realiseert (onderdelen van) software ☒ B1-K1-W4: Test software ☒ B1-K1-W5: Doet verbetervoorstellen voor de software ☐ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☐ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

Introductie

Bij het realiseren van de applicatie via de userstories en het stap voor stap voor stap uitvoeren van de taken is het belangrijk dat je zo mogelijk per taak je werk commit en zodoende een nette historie krijgt van je repository die volledig parallel loopt met je userstories en taken.

Op welke manier moet je commits voorzien van commentaar en hoe vaak dien je te committen?

Tijdens het examen word je o.a. beoordeeld op:

Verzorging code	***De code is verzorgd, leesbaar, gestructureerd en voorzien van zinvol commentaar.***			
	☐ Niet of nauwelijks	☐ Enigszins	☐ Grotendeels	☐ Volledig
Versiebeheer	***Versiebeheer is effectief toegepast.***			
	☐ Niet of nauwelijks	☐ Enigszins	☐ Grotendeels	☐ Volledig

Inhoud

Repository

Na het vastleggen van de product backlog en de eerste sprint met alle userstories en taken begin je met het clonen van je repository. Clonen van je repository doe je slechts eenmaal en ben je er bewust waar de lokale kopie (=**clone**) zich fysiek bevindt. Je kunt het altijd terugvinden omdat in de root van een lokale git repository altijd een .git directory wordt aangemaakt, waarin een soort database wordt aangemaakt die bijhoudt wat de status is van de repository en wat er is veranderd.

.gitignore

Bij het starten van een repository is het belangrijk dat je meteen een goede .gitignore bestand ook toevoegt aan je repository die ervoor zorgt dat niet alles zomaar in je repository wordt toegevoegd.

Sommige bestanden/directories wil je niet in je repository meenemen omdat die alleen lokaal van belang zijn en niet voor anderen van belang zijn.

Voorbeeld bestanden die in een .gitignore worden toegevoegd zijn:

- caches, zoals de inhoud van /node_modules of /packages
- gecompileerde code, zoals .o-, .pyc- en .class-bestanden;
- build-uitvoermappen, zoals /bin, /out of /debug /obj
- bestanden die tijdens runtime zijn gegenereerd, zoals.log, .lock, of .tmp;
- verborgen systeembestanden, zoals .DS_Store of Thumbs.db;

- persoonlijke IDE-configuratiebestanden, zoals .idea/workspace.xml.

Een .gitignore is afhankelijk van wat voor programmeertaal en IDE je gebruikt.

Voor een csharp project is een voorbeeld .gitignore bestand hier te downloaden die je via gitignore.io kunt laten aanmaken.

Hoe negeer je bestanden?

Je plaatst bestandsnamen en/of mapnamen in combinatie met wildcards die ervoor zorgen dat je met 1 regel meerdere matches krijgt die genegeerd gaan worden in de repository.

Patroon	**Voorbeeld matches**	**Uitleg***
**/logs	logs/debug.log logs/monday/foo.bar build/logs/debug.log	met een dubbele asterisk geef je aan dat je alle mappen overal in de repository wilt meenemen
**/logs/debug.log	logs/debug.log build/logs/debug.log **uitgezonderd** logs/build/debug.log	You can also use a double asterisk to match files based on their name and the name of their parent directory.
*.log	debug.log foo.log .log logs/debug.log	Je kunt ook een dubbele asterisk gebruiken om bestanden te matchen op basis van hun naam en de naam van hun bovenliggende map.
*.log !important.log	debug.log trace.log **uitgezonderd** important.log logs/important.log	Als u een uitroepteken voor een patroon plaatst, bedoel je juist dat het niet wordt genegeerd. Als een bestand overeenkomt met een patroon, maar ook met een negatiepatroon dat later in het bestand is gedefinieerd, wordt het niet genegeerd.
.log !important/.log trace.*	debug.log important/trace.log **uitgezonderd** important/debug.log	Patronen die zijn gedefinieerd na een negatiepatroon zullen alle eerder genegeerde bestanden opnieuw negeren.
/debug.log	debug.log ***uitgezonderd*** logs/debug.log	Begin je met een slash dan wordt alleen in de root dit bestand genegeerd
debug.log	debug.log logs/debug.log	Alle debug.log bestanden in iedere map wordt genegeerd
debug?.log	debug0.log debugg.log	Een vraagteken komt precies overeen met één teken.

Patroon	Voorbeeld matches	Uitleg*
	uitgezonderd debug10.log	

* deze uitleg gaat ervan uit dat je .gitignore-bestand zich in de map op het hoogste niveau van je repository bevindt.

Naast deze tekens kun je # gebruiken om commentaar in je .gitignore-bestand op te nemen.

 bron: Atlassian over gitignore

Commits en commit messages?

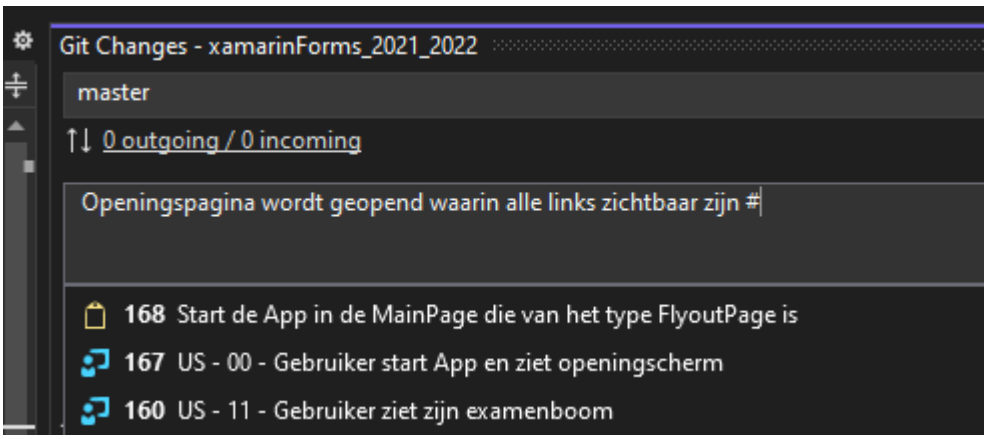
Iedere kleine programmeertaak die je hebt gedaan en hebt getest, waarin geen compileerfouten of executiefoutmeldingen ontstaan moet je committen. Zo kan iedere commit gekoppeld direct of indirect gekoppeld worden aan een userstory of taak.

Voorbeeld in een Xamarin.forms Android App:

In de "**167** US – 00 – Gebruiker opent de App en ziet het openingsscherm" zijn er verschillende programmeertaken te definiëren.

168. Start de App in de MainPage die van het type FlyoutPage is
169. Zet alle links met juiste icons en teksten in de FlyoutPage
170. Maak een FlyoutPage en maak alle losse Contentpages
171. Programmeer de button event zodat de losse pagina's geopend worden

Als je de MainPage klaar hebt en je hebt de App zover dat ie start in de FlyoutPage met de eerste pagina op de achtergrond kun je je een commit doen met de message:



Figuur 1, Als je een # typt verschijnen alle userstories en taken die gekoppeld zijn aan het project

De commit message wordt dan uiteindelijk:

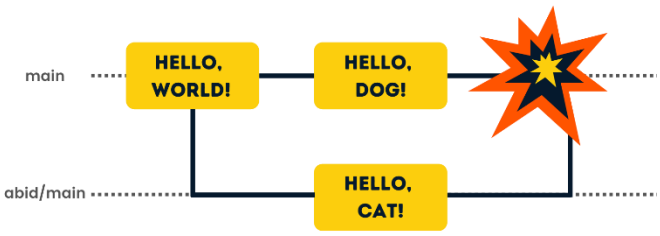
Openingspagina wordt geopend waarin alle links zichtbaar zijn #168 #169 #170

📖 Wil je je verder verdiepen in git? Volg deze cursus op linked in learning.

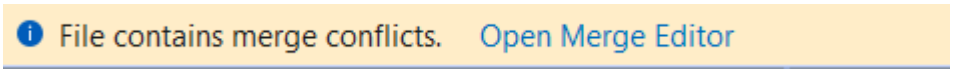
📖 Ben je al een git guru? Volg deze git challenges workshop

Merge conflicts

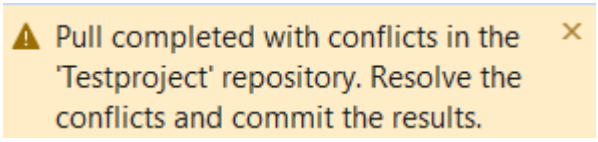
Een merge-conflict ontstaat wanneer twee of meer personen wijzigingen aanbrengen in hetzelfde bestand op verschillende branches en vervolgens proberen deze branches samen te voegen. Azure DevOps kan de wijzigingen niet automatisch samenvoegen omdat het onduidelijk is welke versie behouden moet worden.



Als je wilt jouw wijzigingen wil committen en pushen krijg je een melding dat dit niet kan. Je kan dan kiezen voor pull before push (dus eerst de versie van de server halen en dan alles naar de server sturen. Echter als je met meer dan 1 persoon wijzigingen in een bestand hebt aangebracht krijg je dit boven een file in Visual studio:

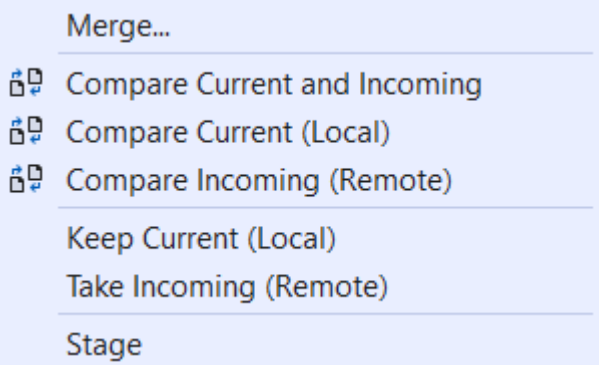


In je git changes tabblad verschijnt deze tekst:

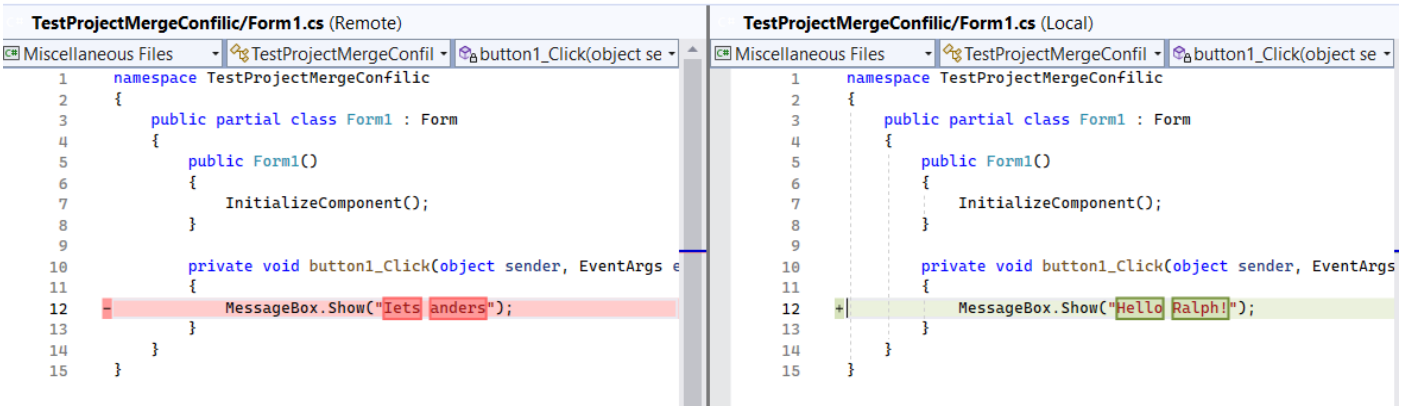


Ofwel het is zaak om je wijzigingen handmatig samen te voegen. Jij (SAMEN MET JE TEAM) bepaalt dan welke code je wilt bewaren.

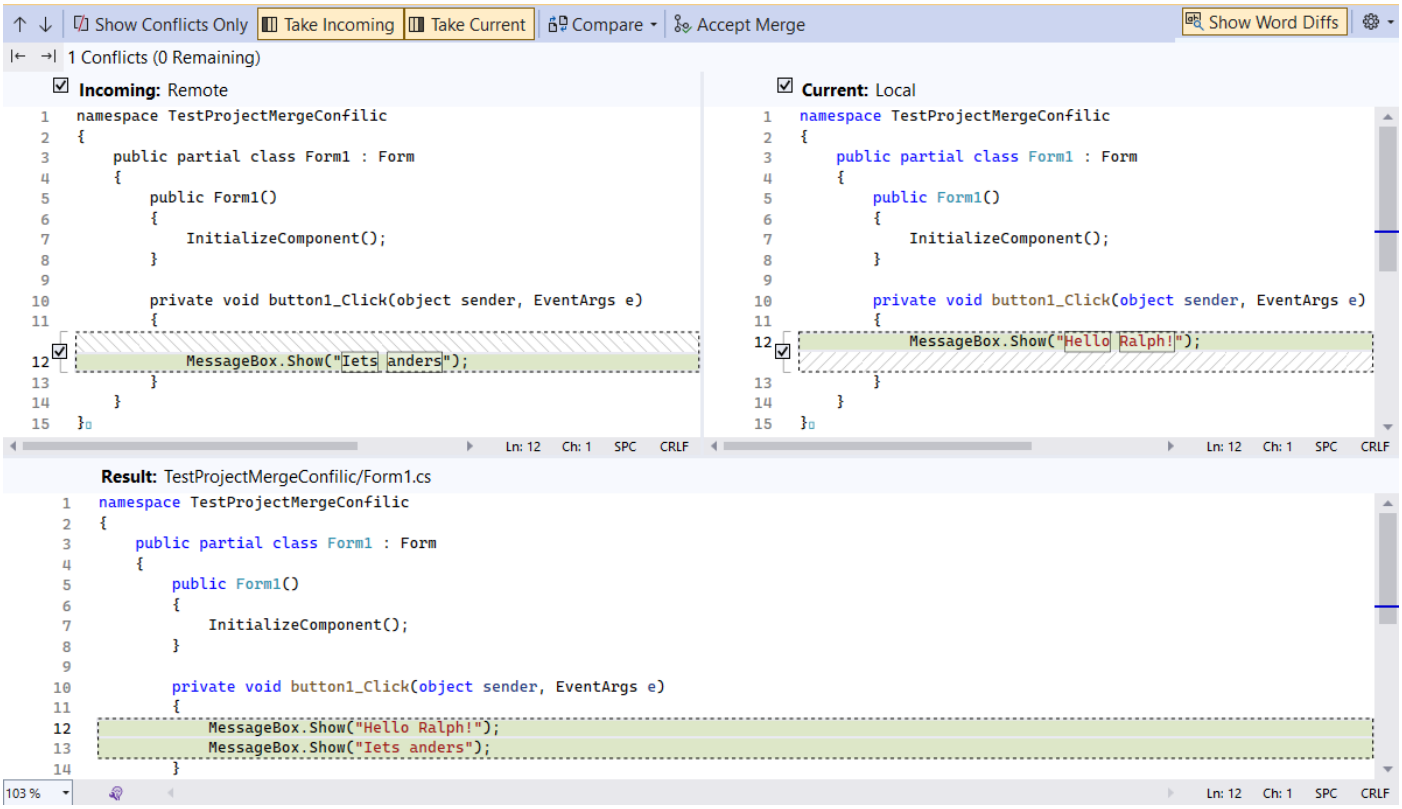
In je git changes kun je per bestand het volgende contextmenu oproepen:



Belangrijke opties hierin zijn natuurlijk het vergelijken van jou versie (current) met degene op de server (incoming, je bent immers aan het pullen als het verkeerd gaat…).



Een andere optie die je kunt toepassen is de Merge Editor. Je krijgt dan onderstaand scherm te zien. Het mooie aan dit scherm is dat je hierbij de mogelijkheid heb om jouw aanpassingen (Take Current) EN de aanpassingen (Take Incomming) van een ander teamlid allebei zou kunnen behouden. Je krijgt in het scherm onderin dan de code die te zien die het uiteindelijk wordt.



Merge-conflicten kunnen enkele uitdagingen met zich meebrengen tijdens het ontwikkelproces. Hier zijn enkele veelvoorkomende uitdagingen waarmee je te maken kunt krijgen bij het oplossen van merge-conflicten:

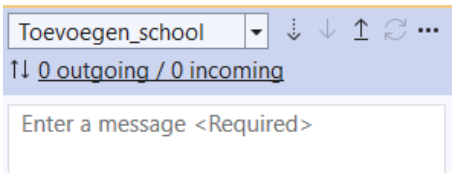
1. **Complexiteit van conflicten**: Merge-conflicten kunnen complex zijn, vooral wanneer er meerdere conflicten optreden in verschillende delen van een bestand of wanneer meerdere bestanden conflicterende wijzigingen hebben. Het vereist vaak een grondige analyse en begrip van de wijzigingen om de beste oplossing te vinden.
2. **Afhankelijkheden en impact**: In complexe projecten kunnen conflicten in één bestand gevolgen hebben voor andere delen in je project. Het oplossen van een merge-conflict in een bestand kan dus invloed hebben op de functionaliteit van andere delen van het project. Dit vereist zorgvuldige aandacht om ervoor te zorgen dat de oplossing geen onbedoelde neveneffecten heeft.
3. **Tijdsdruk**: Merge-conflicten kunnen vertragingen veroorzaken in het ontwikkelproces, vooral als ze complex zijn en extra tijd en inspanning vergen om op te lossen. Dit kan de deadlines van het project beïnvloeden en druk leggen op het ontwikkelteam.
4. **Communicatie en samenwerking**: Merge-conflicten komen vaak voor in teams waar meerdere ontwikkelaars aan hetzelfde project werken. Het effectief communiceren en samenwerken om conflicten op te lossen is essentieel. Het kan uitdagend zijn om duidelijkheid te krijgen over de bedoelingen van verschillende ontwikkelaars en om tot een consensus te komen over de beste oplossing.
5. **Complexe projecten**: In grote projecten met veel bestanden en branches kunnen merge-conflicten complexer worden. Het kan uitdagend zijn om een goed overzicht te krijgen van alle conflicten en om te begrijpen hoe de wijzigingen in verschillende delen van het project met elkaar samenhangen.

Om deze uitdagingen aan te pakken, is het belangrijk om een gestructureerde aanpak te volgen bij het oplossen van merge-conflicten. Dit omvat het begrijpen van de wijzigingen, communiceren met andere ontwikkelaars, het zorgvuldig analyseren van de impact van conflicten en het zorgvuldig testen van de oplossingen om ervoor te zorgen dat er geen onbedoelde gevolgen zijn.

Werken met verschillende brances

Het werken met aparte branches in Azure DevOps en Visual Studio 2022 stelt ontwikkelaars in staat om parallel aan het hoofdproject te werken zonder de hoofdcode (master brance) te beïnvloeden. Hier zijn de stappen om in aparte branches te werken:

1. Branch maken: Open Visual Studio 2022 en ga naar het "Git Changes" venster. Klik op "..." en vervolgens op "New Branch". Geef de nieuwe branch een naam die relevant is voor jouw werk, bijvoorbeeld een specifieke functie of bugfix.
2. Switchen naar de nieuwe branch: In het Git Changes kun je kiezen in welke branche je werkt:



3. Wijzigingen aanbrengen: Maak wijzigingen in de code, voeg nieuwe functies toe, pas bestaande code aan, etc. in Visual Studio 2022.

4. Commit wijzigingen: Zodra je tevreden bent met jouw wijzigingen, gebruik je het "Git Changes" venster om de wijzigingen te committen naar jouw lokale branch. Geef een beschrijvende commit-boodschap om de aard van de wijzigingen aan te geven.

5. Push naar de remote repository: Om jouw lokale branch naar de remote repository te pushen, klik je op "Sync" in het "Team Explorer" venster en vervolgens op "Push". Hierdoor worden jouw wijzigingen gedeeld met andere ontwikkelaars die toegang hebben tot de repository.

Voor- en nadelen van werken met aparte branches:

Voordelen:

- **Parallelle ontwikkeling**: Aparte branches stellen ontwikkelaars in staat om tegelijkertijd aan verschillende functies of bugfixes te werken zonder elkaar in de weg te zitten.
- **Experimentatie**: Je kunt nieuwe ideeën en experimenten uitvoeren in een aparte branch zonder de stabiele hoofdcodebase te verstoren.
- **Gecontroleerde releases**: Je kunt aparte branches gebruiken om nieuwe functies te ontwikkelen en testen voordat ze worden samengevoegd met de master branch, waardoor een betere controle over releases ontstaat.

Nadelen:

- **Merge-conflicten**: Bij het samenvoegen van branches kunnen merge-conflicten ontstaan als meerdere ontwikkelaars wijzigingen hebben aangebracht in hetzelfde deel van de code. Deze conflicten moeten handmatig worden opgelost.
- **Overhead**: Werken met meerdere branches kan extra beheer- en coördinatiewerk met zich meebrengen, omdat ontwikkelaars moeten afstemmen welke wijzigingen wanneer worden samengevoegd met de hoofdbranch.
- **Complexiteit**: Het werken met meerdere branches kan de complexiteit van het ontwikkelproces vergroten, vooral als er veel branches zijn en de afhankelijkheden tussen branches ingewikkeld zijn.