Avans DevOps

Software Ontwerp & Architectuur 3

Mick Holster, Juliet van BEzooijen (2183903)

2024

Inhoudsopgave

[Functionele & non-functionele requirements 2](#_Toc162214629)

[Functionele Requirements 2](#_Toc162214630)

[Non-Functionele Requirements 4](#_Toc162214631)

[Design Patterns 5](#_Toc162214632)

[Creational Patterns 5](#_Toc162214633)

[Factory pattern 5](#_Toc162214634)

[Structural patterns 6](#_Toc162214635)

[Adapter pattern 6](#_Toc162214636)

[Composite Pattern 6](#_Toc162214637)

[Behavioural Pattern 7](#_Toc162214638)

[Strategy pattern 7](#_Toc162214639)

[Observer pattern 8](#_Toc162214640)

[State pattern 9](#_Toc162214641)

[Template pattern 10](#_Toc162214642)

[Testen 10](#_Toc162214643)

# Functionele & non-functionele requirements

## Functionele Requirements

Projectmanagement:

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | User story |  | Acceptatiecriteria |
| AD-1 | Als een Product Owner wil ik een nieuw project kunnen aanmaken, zodat mijn team aan de slag kan met een nieuwe opdracht. |  | * Project kan aangemaakt worden * Naam, beschrijving, en startdatum kunnen toegevoegd worden |
| AD-2 | Als een Scrum Master wil ik sprints kunnen plannen, zodat het team weet wat er op de planning staat voor de komende sprints. |  | * Een sprint kan aangemaakt worden * Een sprint bevat een specifieke begin- en einddatum en een status * De sprint ***status*** verandert naar ‘Created’ wanneer deze is aangemaakt. * Een sprint kan aan een project toegevoegd worden |
| AD-3 | Als een Scrum Master wil ik een type aan een sprint kunnen toevoegen, zodat ik makkelijk kan zien of er een review plaats moet vinden of niet |  | * Als de sprint van type ‘review’ is moet er een samenvatting van de review geüpload kunnen worden na de release. |
| AD-4 | Als een Developer wil ik taken kunnen toevoegen aan de product backlog, zodat we een overzicht hebben van alle werkzaamheden. |  | * Nieuwe backlog items kunnen aangemaakt worden * De ***status*** van het item verandert naar ‘ToDo’ wanneer deze is aangemaakt. * Backlog items kunnen toegevoegd worden aan een (nog niet actieve) sprint * Een backlog item bevat een titel, huidige status, beschrijving, en een aantal story points * Een backlog item kan gekoppeld worden aan 1 user |
| AD-5 | Als een Developer wil ik activiteiten kunnen toevoegen aan een backlog item, zodat meerdere developers aan 1 backlog item kunnen werken. |  | * Activiteiten kunnen aangemaakt worden * Een activiteit bevat een titel * Er kan een user aan een activiteit toegewezen worden |
| AD-6 | Als een developer wil k een forum kunnen toevoegen aan een backlogitem, zodat er vragen gesteld kunnen worden over verschillende onderwerpen. |  | * Een forum kan aangemaakt worden * Een forum bevat een titel, huidige status, backlogitem en een titel * Een forum krijgt de ***status*** ‘Open’ wanneer deze is aangemaakt. |
| AD-7 | Als een gebruiker wil ik berichten kunnen posten op een specifiek forum, zodat ik vragen kan stellen aan mijn collega’s. |  | * Er kan een bericht toegevoegd worden aan een forum * Er kan een gereageerd worden op een bericht met een ander bericht |

Status (sprint, backlog, forum):

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | User story |  | Acceptatiecriteria |
| AD-8 | Als een Scrum Master wil ik sprints kunnen starten, zodat het team kan beginnen met werken aan de doelen voor die sprint. |  | * De sprint status verandert naar ‘In progress’ wanneer deze is gestart. * Backlog items kunnen niet meer toegevoegd worden * Informatie van de sprint kan niet meer veranderd worden |
| AD-9 | Als een Scrum Master wil ik een sprint kunnen markeren als af, zodat we de sprint kunnen evalueren. |  | * De sprint status verandert naar ‘Finished’ wanneer deze is klaar is. * Backlog items kunnen niet meer toegevoegd worden |
| AD-10 | Als een Scrum Master wil ik een sprint kunnen releasen, zodat de functies in de productieomgeving terechtkomen. |  | * Na een review kan de sprint gereleased worden. * Release pipeline wordt gestart op basis van sprint type (review of niet) * De sprint krijgt de status ‘Releasing’ * Wanneer alles goed uitgevoerd is krijgt de sprint de status ‘Released’ en krijgen de scrum master en product owner een ***notificatie*** * Wanneer er iets fout gaat krijgt scrum master een ***notificatie*** |
| AD-11 | Als een developer, wil ik controleren of een backlog item voldoet aan de Definition of Done voordat het als klaar gemarkeerd wordt, zodat de kwaliteit van de code gewaarborgd blijft. |  | * Als een item de status ‘Tested’ heeft, kan een developer het gaan vergelijken met de Definition of Done. * Als het item niet aan alle criteria voldoet, krijgt het de status "Ready for Testing". * Als het item aan alle criteria voldoet, krijgt het de status “Done” |
| AD-12 | Als een teamlid, wil ik dat het forum voor een backlog item automatisch gesloten wordt wanneer het item de "Done" status bereikt, zodat het duidelijk dit item afgerond is. |  | * Zodra een backlog item status ‘Done’ heeft, worden de bijbehorende forums gesloten voor nieuwe reacties. |

Notificaties:

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | User story |  | Acceptatiecriteria |
| AD-13 | Als een Tester, wil ik een notificatie ontvangen wanneer een backlog item klaar is voor testing, zodat ik onmiddellijk met testen kan beginnen. |  | * Een automatische notificatie wordt verzonden naar de testers zodra een backlog item naar "Ready for Testing" wordt verplaatst. |
| AD-14 | Als een Scrum Master, wil ik een notificatie ontvangen wanneer een backlog item terug naar "Todo" moet vanwege fouten of ontbrekende elementen tijdens de "Testing" fase, zodat ik kan zorgen dat de problemen worden aangepakt. |  | * Een automatische notificatie wordt verzonden naar de Scrum Master wanneer een item teruggaat naar "Todo". |
| AD-15 | Als een Teamlid, wil ik een notificatie ontvangen wanneer iemand een nieuw bericht plaatst in een forum, zodat ik op de hoogte blijf van de laatste discussies. |  | * Een automatische notificatie wordt verzonden naar alle teamleden wanneer een nieuw bericht wordt geplaatst in een forum. |

Rapport:

|  |  |  |  |
| --- | --- | --- | --- |
| Nr | User story |  | Acceptatiecriteria |
| AD-16 | Als een Scrum Master, wil ik na het sluiten van een sprint automatisch een sprint rapportage kunnen genereren, zodat er een overzicht is van wat er is bereikt en wat de volgende stappen zijn. |  | * Een sprint rapport wordt automatisch gegenereerd zodra de sprint de ***status*** "Closed" bereikt. * Teamleden kunnen het rapport in diverse formaten (PDF, PNG) downloaden. |
| AD-17 | Als een Scrum Master, wil ik een samenvatting van de sprint review toe kunnen voegen aan het sprint rapport, zodat stakeholders inzicht krijgen in de feedback en beslissingen die tijdens de review zijn genomen. |  | * De Scrum Master kan een tekstuele samenvatting uploaden die aan het sprint rapport wordt toegevoegd. |

## Non-Functionele Requirements

|  |  |
| --- | --- |
| Nr | User story |
| AD-18 | Als een Developer, wil ik dat mijn code automatisch wordt beoordeeld door SonarCloud, zodat ik kan controleren dat het een Quality Gate label A heeft, wat betekent dat de code van hoge kwaliteit is. |
| AD-19 | Als een Developer, wil ik unit tests implementeren voor de logica in het domein, zodat we kunnen garanderen dat de applicatie betrouwbaar werkt en makkelijk te onderhouden is. Dit wordt gedaan door de code coverage in SonarCloud zo hoog mogelijk te krijgen. |
| AD-20 | Als een Developer, wil ik gebruikmaken van stub-implementaties en mocking voor externe afhankelijkheden in mijn unit tests, zodat de code getest kan worden zonder afhankelijk te zijn van externe systemen. |
| AD-21 | Als een Systeemarchitect, wil ik meerdere design patterns implementeren, zodat het systeemontwerp flexibel en uitbreidbaar is, om gemakkelijk nieuwe varianten van concepten te kunnen integreren zonder de bestaande functionaliteiten aan te passen. |

# Design Patterns

## Creational Patterns

### Factory pattern

Het gebruik van het factory pattern in deze casus biedt een flexibele en uitbreidbare manier om objecten te creëren zonder de exacte klassen van objecten die moeten worden gecreëerd te specificeren. Er is gekozen om dit pattern te implementeren om de rollen van een user in het systeem te creëren en toe te voegen aan een user. De UserFactory (zie Figuur 1) kan gebruikers met verschillende rollen (zoals Developer, Product Owner, Scrum Master, en Tester) te creëren met minimale koppeling tussen de code die de gebruikersinstanties maakt en de specifieke rolklassen.

A screenshot of a computer screen

Description automatically generated

Figuur 1: UML implementatie factory pattern

Waarom factory pattern?

1. **Flexibiliteit in creatie van objecten**: Het maakt de creatie van user objecten met verschillende rollen mogelijk zonder dat de code die de user aanmaakt directe afhankelijkheden heeft op de verschillende klassen. Dit bevordert *loose coupling* en maakt het systeem flexibeler.
2. **Uitbreidbaarheid**: Nieuwe rollen kunnen gemakkelijk worden toegevoegd zonder de bestaande code in de UserFactory te wijzigen. Zolang de nieuwe rollen de IRole interface implementeren, kunnen ze dynamisch worden geladen en geïnstantieerd gebaseerd naam van de rol.
3. **Verminderen redundantie**: Het centraliseert de logica voor het instantiëren van gebruikers en hun rollen, wat redundantie vermindert en een enkel punt van onderhoud voor objectcreatie biedt.

## Structural patterns

### Adapter pattern

Het Adapter Pattern wordt gebruikt om een standaard interface (INotificationAdapter**,** zie Figuur 2) te bieden voor verschillende notificatiediensten (zoals Slack, Teams, en Email), waardoor de rest van het systeem op een consistente wijze notificaties kan versturen zonder zich zorgen te maken over de specifieke implementaties van elke service. Dit pattern maakt het systeem flexibeler en onderhoudsvriendelijker door de koppeling tussen het notificatiesysteem en de externe notificatiediensten te verminderen.

A screenshot of a computer screen

Description automatically generated

Figuur 2: UML implementatie adapter pattern

Waarom Adapter Pattern?

1. **Makkelijke integratie**: Door alle externe notificatiediensten te voorzien van dezelfde interface, kan het systeem notificaties versturen op 1 manier. Dit vereenvoudigt de integratie van nieuwe notificatiediensten in de toekomst. Het systeem hoeft niet direct te communiceren met de API's van externe diensten, wat de complexiteit vermindert. In plaats daarvan spreekt het systeem met de adapter, die zorgt voor de communicatie met de service die communiceert met de externe dienst.
2. **Uitbreidbaarheid:** Nieuwe notificatiediensten kunnen gemakkelijk worden toegevoegd door nieuwe services toe te voegen zonder de logica van het systeem aan te passen.
3. **Wijzigingen**: Als er iets in de notificatiedienst wijzigt (bijv. een wijziging in de API), hoeft alleen die specifieke service aangepast te worden. De rest van het systeem blijft ongewijzigd, wat de impact van wijzigingen tot een minimum houdt.

### Composite Pattern

**Composite Pattern**: Helpt bij het beheren van een groep objecten op dezelfde manier als een enkel object. Dit is nuttig voor het modelleren van de hiërarchische structuur van projecten, waarbij projecten, sprints, en backlog items als een eenheid of afzonderlijk kunnen worden behandeld.

Het Composite Pattern wordt gebruikt binnen een forum om een hiërarchische structuur van berichten te beheren. Dit stelt het systeem in staat om zowel individuele berichten (MessageLeaf, zie Figuur 3) als reacties op berichten (composites) op dezelfde manier te behandelen.

A screenshot of a computer program

Description automatically generated

Figuur 3: UML implementatie composite pattern

Waarom Composite Pattern?

1. **Consistentie**: Het Composite Pattern maakt het mogelijk om individuele objecten en geneste vormen van objecten hetzelfde te behandelen. In het forum kunnen zowel normale berichten als threads van berichten (reacties op een ander bericht) voorkomen, die allemaal weergeven moeten worden op dezelfde manier.
2. **Vereenvoudiging**: Door gebruik te maken van het Composite Pattern, wordt de code voor het beheren van berichten vereenvoudigd. De implementatie van de hoeft niet te verschillen tussen enkele berichten en geneste berichten (kan wel), waardoor de code eenvoudiger en onderhoudbaar wordt.

## Behavioural Pattern

### Strategy pattern

Het Strategy Pattern wordt toegepast om flexibel verschillende exportstrategieën te kunnen gebruiken voor het exporteren van rapporten. Dit patroon maakt het mogelijk om het exporteren van rapporten dynamisch te selecteren op runtime, gebaseerd op de behoeften van de specifieke user. Hierdoor kan de applicatie eenvoudig uitgebreid worden met nieuwe exportformaten zonder bestaande code aan te passen. Dit bevordert *open/closed-principe* van SOLID ontwerpprincipes, omdat de klassen open staan voor uitbreiding maar niet gewijzigd hoeven te worden.

A screenshot of a computer

Description automatically generated

Figuur 4: UML implementatie strategy pattern

Waarom Strategy Pattern?

1. **Flexibiliteit**: Het Strategy Pattern biedt flexibiliteit door verschillende strategieën te instantiëren met dezelfde interface. Dit maakt het makkelijk om het gedrag van een object te veranderen zonder de code aan te passen door een andere strategie te gebruiken.
2. **Uitbreidbaarheid**: Nieuwe exportformaten kunnen eenvoudig toegevoegd worden door nieuwe strategieën te implementeren, zonder de bestaande code te wijzigen.
3. **Seperation of concerns**: Het patroon scheidt de implementatie van de rest van het systeem. Hierdoor is de code beter georganiseerd en maakt het makkelijker om individuele strategieën te testen en te debuggen.

### Observer pattern

Het Observer Pattern wordt toegepast om relatie te creëren waarbij meerdere observers (User, zie Figuur 5) automatisch een notificatie krijgen wanneer de status van één subject (Sprint, BacklogItem of Forum) verandert. Dit patroon is handig om te gebruiken in een systeem waar de componenten invloed op elkaar kunnen hebben, maar niet *strongly coupled* hoeven te zijn. In dit geval krijgt een User dus updates over de status van een sprint, backlog item of forum waar hij/zij aan meedoet. Ook wordt er in deze implementatie rekening gehouden met de rol van een gebruiker. Het systeem kan hierdoor ook specifiek berichten sturen naar bijvoorbeeld alleen een scrum master of een product owner.

A screenshot of a diagram

Description automatically generated

Figuur 5: UML implementatie observer pattern

Waarom het Observer Pattern?

1. **Decoupling**: Het Observer Pattern maakt *loose coupling* mogelijk tussen subjects en observers. Subjects weten niet welke specifieke observers een notificatie moeten krijgen, alleen dat ze een lijst met observers hebben geüpdatet moeten worden.
2. **Uitbreidbaarheid**: Nieuwe observers kunnen zich makkelijk abboneren/registreren op updates van subjects en bestaande observers kunnen zich uitschrijven. Dit maakt het systeem flexibel en uitbreidbaar.

### State pattern

Het State Pattern wordt gebruikt om de verschillende toestanden (states) van backlog items, forums en sprints te beheren. Dit patroon maakt het mogelijk om het gedrag van een object te veranderen wanneer zijn toestand wijzigt. Dit is handig voor objecten die verschillende fasen kunnen hebben, zoals de onderstaande cycli van de sprint, backlog item en forum objecten.

**Sprint States**:

* + Created
  + In progress
  + Finished
  + Releasing
  + Released
  + Closed
  + Cancelled

**Backlog States:**

* + Todo
  + Doing
  + Testing
  + Tested
  + Done

**A graph paper with a rectangular piece of paper

Description automatically generated**

Figuur 6: UML implementatie state pattern

**Forum States:**

* + Open
  + Closed

Waarom State Pattern?

1. **Toestandspecifieke logica**: Het State Pattern stelt objecten in staat hun gedrag te wijzigen wanneer hun interne toestand verandert. Dit wordt gedefinieerd in aparte klassen (zie Figuur 6). Dit maakt het systeem makkelijker te begrijpen en te onderhouden, omdat de logica voor elke staat apart gedefinieerd is.
2. **Minder condities**: Door het gebruik van het State Pattern worden er minder conditionnele statements (if-else of switch statements) geschreven. Dit maakt de code overzichtelijker en minder foutgevoelig.

### Template pattern

Het Template Pattern wordt scenario gebruikt om een ontwikkelingspipeline in een DevOps omgeving na te bootsen. Dit patroon zorgt ervoor dat een algoritme in een methode van een superclass te gedefinieerd kan worden, maar laat subclasses bepaalde stappen van het algoritme te wijzigen of dingen toe te voegen zonder de structuur van het algemene algoritme te veranderen. Het is toegepast om de samenvatting van een sprint review toe te kunnen voegen aan de release pipeline. Dit gebeurt namelijk niet als een sprint geen sprint review heeft, maar wel als de review wel plaatsvindt en de sprint dus van het type ‘Review’ is.

A screenshot of a diagram

Description automatically generated

Figuur 7: UML implementatie template pattern

Waarom Template Pattern?

1. **Skelet van een workflow**: Het Template Pattern zorgt ervoor dat een algemeen skelet van een workflow gedefinieerd kan worden. Dit skelet kan vervolgens ook gebruikt worden in subclasses. Dit kan handig zijn voor het opzetten van een proces waarbij specifieke stappen variabel kunnen zijn.
2. **Herbruikbaarheid**: Door de algemene structuur van de pipeline in een superclass (PipelineTemplate) te definiëren, zorgt het patroon voor herbruikbaarheid en een duidelijke structuur binnen van de code. Dit maakt het systeem gemakkelijker te begrijpen en te onderhouden.

# Testen

**Sprints:**

* **AD-8** (Sprints kunnen starten): De tests voor de **InProgressState** controleren of de staat van een sprint correct verandert naar 'In Progress' wanneer deze wordt gestart. Dit valideert de functionaliteit die nodig is om te voldoen aan de requirement dat sprints gestart kunnen worden en dat hun status dienovereenkomstig verandert.
* **AD-9** (Een sprint markeren als af): De tests voor de **FinishedState** bevestigen dat de staat van een sprint correct verandert naar 'Finished' wanneer deze is afgerond, wat aansluit bij de behoefte om sprints te kunnen markeren als voltooid.
* **AD-10** (Een sprint releasen): De tests voor **ReleasingState** en **ReleasedState** valideren de overgangen naar en van de 'Releasing' en 'Released' staten, wat essentieel is voor het releasen van sprints conform de beschreven user story.

**Backlog Items:**

* **AD-4** (Taken toevoegen aan de product backlog): De initiële staat van een backlog item wordt getest met **TodoState** tests, wat aansluit bij de requirement dat nieuwe items als 'ToDo' gemarkeerd moeten worden.
* **AD-11** (Controleren of een backlog item voldoet aan de Definition of Done): De **TestedState** en **DoneState** tests controleren of items correct overgaan naar 'Done' wanneer ze voldoen aan de vereiste criteria, en of ze correct kunnen terugkeren naar een vorige staat als dat niet het geval is.

Om de cycli van een backlog item te testen wordt er gebruik gemaakt van een graaf. Deze graaf geeft overzichtelijk weer welke routes een backlog item allemaal kan nemen om uiteindelijk afgerond te worden. Door deze graaf te maken is er een duidelijk beeld van welke testroutes allemaal geschreven moeten worden.

A diagram of a test

Description automatically generated

Figuur 8: Graaf backlog items status

**Non-Functionele Requirements**

**AD-18: Automatische Codebeoordeling met SonarCloud**

* Requirement: Als Developer wil ik dat mijn code automatisch wordt beoordeeld door SonarCloud zodat ik kan controleren dat het een Quality Gate label A heeft, wat betekent dat de code van hoge kwaliteit is.
* Tests: Hoewel het document niet expliciet testscenario's voor AD-18 beschrijft, impliceert de aanwezigheid van deze requirement de implementatie van CI/CD pipelines die geïntegreerd zijn met SonarCloud voor het automatisch beoordelen van de codekwaliteit bij elke push naar de repository. Dit zorgt ervoor dat de code consistent voldoet aan vastgestelde kwaliteitsnormen.

**AD-19: Implementatie van Unit Tests**

* Requirement: Als Developer wil ik unit tests implementeren voor de logica in het domein zodat we kunnen garanderen dat de applicatie betrouwbaar werkt en makkelijk te onderhouden is. Dit wordt gedaan door de code coverage in SonarCloud zo hoog mogelijk te krijgen.
* Tests: Specifieke tests, waaronder TodoState tests voor backlog items en de diverse state tests voor sprints, demonstreren de directe toepassing van unit testing. Deze tests zijn ontworpen om de correcte functionaliteit van verschillende systeemcomponenten te verifiëren en om te zorgen dat de codebase voldoet aan de Definition of Done, waarbij betrouwbaarheid en onderhoudbaarheid vooropstaan.

**Algemene Observaties:**

* **AD-20** (Gebruik van stub-implementaties en mocking in unit tests): Alle tests maken gebruik van mocking frameworks zoals NSubstitute, wat overeenkomt met de wens om externe afhankelijkheden in tests te isoleren. Dit zorgt voor betrouwbaarheid en onderhoudbaarheid van de code.
* **AD-21** (Implementatie van meerdere design patterns): De toepassing van het State pattern wordt direct getest in de context van zowel sprints als backlog items. Dit toont aan dat het systeemontwerp flexibel en uitbreidbaar is, wat overeenkomt met de non-functionele requirements.

De ontwikkelde unit tests bieden een duidelijke controle van zowel functionele als non-functionele requirements. Ze dragen bij aan het vertrouwen dat het systeem correct functioneert volgens de gespecificeerde user stories en requirements. Het uitgebreide gebruik van design patterns en best practices in de zorgt voor een goede systeemarchitectuur die flexibiliteit, uitbreidbaarheid en onderhoudbaarheid bevordert.

Top of Form