

Software Quality

GEBASSEERD OP MVCLIBRARYAPP

Robin Kantier & Quinten Schaap
STUDENTEN WINDESHEIM ALMERE, ADSD1.

Inhoud

1.	Inleiding	3
2.	Definities.....	4
2.1	Kwaliteit	4
2.2	Software.....	4
2.3	Kwaliteit in software.....	4
3.	Verschil tussen Algemene Kwaliteit en Software Kwaliteit.....	5
3.1	Algemene Kwaliteit	5
3.2	Software Kwaliteit	5
4.	ISO 25010	5
4.1	Inleiding en Begrip van ISO 25010	5
4.2	Belang van ISO 25010	5
4.3	De Acht Kenmerken van ISO 25010	6
4.3.1	Beschrijving en Locatie in de Software Ontwikkelingscyclus	6
4.3.2	Hoe elke Hoofdkenmerk zichtbaar wordt in de Software Ontwikkelingscyclus	7
5.	ISO 25040	8
5.1	Begrip van ISO 25040	8
5.2	ISO 25040 voor een Software Ontwikkelaar	8
5.3	De Onderdelen van ISO 25040	8
5.4	Het Gebruik van ISO 25040 in de Software Ontwikkelingscyclus.....	8
6.	Minimum Viable Product (MVP)	9
6.1	Begrip van MVP	9
6.2	Voor- en Nadelen van een MVP.....	9
7.	ISO 9000 en 9001.....	10
7.1	Begrip van ISO 9000 en 9001	10
7.2	Quality Management Principles (QMP's) van ISO 9001.....	10
7.3	Voor- en Nadelen van ISO 9000 en 9001	10
8.	Functional en Non-Functional Requirements	11
8.1	Begrip van Functional Requirements	11
8.2	Begrip van Non-Functional Requirements	11
8.3	Voorbeeldproject.....	11
8.3.1	Vijf Functional Requirements en Testplan.....	11
8.3.2	Vijf Non-Functional Requirements en Testplan	12
9.	Continuous Integration and Continuous Delivery (CI/CD).....	13

9.1 Begrip van CI/CD en CI/CD Pipeline.....	13
9.2 MVCLibraryApp CI/CD Pipeline.....	14
10. Testen.....	15
10.1 Begrip en Soorten van Testen.....	15
10.2 Test Uitwerkingen op MVCLibraryApp	15
10.2.1 UI-test	15
10.2.2 Integratie-test.....	16
10.2.3 Unit-tests.....	16
10.3 Complementaire Natuur van Testen	17
10.4 Implementatie van Testen in een Organisatie.....	18
10.5 Belang van Testen	19
11. Conclusie.....	20
12. Referenties	21
13. Reflectie	22
13.1 Reflectieverslag (Quinten Schaap):	22
13.1.1Feedbackverslag (Quinten Schaap voor Robin Kantier):	22
13.2Reflectieverslag (Robin Kantier):	22
13.2.1 Feedbackverslag (Robin Kantier voor Quinten Schaap):	22

1. Inleiding

In onze steeds meer verbonden wereld speelt software een centrale rol. Van de apps die we elke dag gebruiken op onze smartphones tot complexe systemen die cruciale bedrijfsprocessen ondersteunen, software heeft een immense impact op ons dagelijks leven. Daarom is het begrijpen van de principes van softwareontwikkeling en, belangrijker nog, de kwaliteit van de software die we ontwikkelen, van het grootste belang.

Dit verslag is door ons (Quinten Schaap en Robin Kantier), als duo, samengesteld als onderdeel van onze studie in Software Development, met name voor het vak Software Quality. Wij zullen ons richten op diverse kernconcepten binnen softwareontwikkeling, waaronder de definities en betekenissen van 'kwaliteit' en 'software', het belang en de kenmerken van ISO normen zoals ISO 25010 en ISO 25040, het concept van een Minimum Viable Product (MVP), het onderscheid tussen functionele en niet-functionele eisen, en het proces van continue integratie en levering (CI/CD). Daarnaast zullen we het belang van verschillende testmethoden binnen de softwareontwikkelingscyclus benadrukken, met bijzondere aandacht voor hoe deze tests elkaar kunnen aanvullen en geïmplementeerd kunnen worden binnen een organisatie.

Om deze concepten te illustreren, zullen we een voorbeeldproject, de MVC Library App, die we tijdens onze Software Development 2 cursus hebben ontwikkeld, gebruiken. Dit project zal dienen als een praktische casus om de genoemde concepten in actie te zien en om te demonstreren hoe ze de effectieve ontwikkeling en implementatie van software kunnen bevorderen.

Door dit verslag hopen we een dieper begrip te krijgen van de complexiteit en het belang van kwaliteitsbeheer in softwareontwikkeling. Het is ons doel om onze kennis van deze concepten te demonstreren en te laten zien hoe we ze in praktijk kunnen brengen in een reëel softwareontwikkelingsproject.

2. Definities

2.1 Kwaliteit

'Kwaliteit' is zo'n woord dat je overal tegenkomt, vooral als het gaat om het maken van dingen, of dat nu fysieke producten of software is. Maar wat betekent kwaliteit precies? Nou, in de meest simpele zin, betekent kwaliteit dat iets goed werkt - het doet wat het hoort te doen en het doet het goed. Dit kan variëren van een pen die lekker schrijft, een auto die soepel rijdt, tot een app die zonder haperen werkt.

2.2 Software

'Software' is een ander woord dat we vaak tegenkomen, maar wat betekent het precies? In de eenvoudigste termen, software is wat een computer vertelt wat te doen. Als je een app op je telefoon opent, is dat software aan het werk. Als je een document typt op je computer, gebruik je software. Het is in feite de instructies die we aan een computer geven om een bepaalde taak uit te voeren.

2.3 Kwaliteit in software

Als we deze twee ideeën samenbrengen, krijgen we 'softwarekwaliteit'. Dit gaat over hoe goed een stuk software zijn werk doet. Doet het wat het moet doen? Loopt het soepel? Is het gemakkelijk te gebruiken? Kun je erop vertrouwen dat het niet plotseling zal crashen? Dit zijn de soorten vragen die we stellen als we het hebben over softwarekwaliteit.

In de volgende hoofdstukken zullen we een paar meer formele tools en ideeën bekijken die mensen gebruiken om te praten over en te meten hoe goed software is, zoals de ISO 25010 en ISO 25040 normen. We zullen deze ideeën ook toepassen op onze eigen softwareproject, de MVC Library App, om je een beter idee te geven van hoe dit in de praktijk werkt.

3. Verschil tussen Algemene Kwaliteit en Software Kwaliteit

3.1 Algemene Kwaliteit

Als we het hebben over 'kwaliteit' in het algemeen, denken we meestal aan hoe goed iets werkt, hoe duurzaam het is, hoe goed het is gemaakt, enzovoort. Als je een kwaliteitsauto koopt, verwacht je bijvoorbeeld dat hij soepel rijdt, dat hij niet vaak gerepareerd hoeft te worden, en dat hij er goed uitziet en aanvoelt. Dit zijn allemaal tastbare, meetbare dingen.

3.2 Software Kwaliteit

Bij software wordt kwaliteit een beetje ingewikkelder. Natuurlijk, een deel ervan gaat over dingen die we ook bij andere producten bekijken: doet de software wat het hoort te doen? Is het makkelijk te gebruiken? Is het het geld waard?

Maar er zijn ook extra dingen waar we naar kijken bij softwarekwaliteit. Dingen zoals: hoeveel bugs heeft het? Hoe goed werkt het met andere stukken software? Hoe veilig is het? Is het gebouwd op een manier die het gemakkelijk maakt om in de toekomst wijzigingen aan te brengen?

In het volgende hoofdstuk gaan we dieper in op enkele van de specifieke dingen waar we naar kijken als we het over softwarekwaliteit hebben, met behulp van een systeem genaamd ISO 25010.

4. ISO 25010

4.1 Inleiding en Begrip van ISO 25010

ISO 25010 is een internationale standaard voor softwareproductkwaliteit en kwaliteit in gebruik. Het is een krachtig hulpmiddel dat ons helpt om de kwaliteit van softwareproducten te meten en te verbeteren. Het biedt een uitgebreid overzicht van wat we onder 'kwaliteit' verstaan als het om software gaat.

4.2 Belang van ISO 25010

Het belang van ISO 25010 kan niet worden onderschat. Door deze standaard te volgen, kunnen we zorgen voor een hoge kwaliteit van onze softwareproducten, wat leidt tot een hogere tevredenheid van de gebruiker en een hogere productiviteit. Bovendien kan het ons helpen om problemen vroegtijdig op te sporen en te corrigeren, waardoor de onderhoudskosten worden verlaagd.

4.3 De Acht Kenmerken van ISO 25010

4.3.1 Beschrijving en Locatie in de Software Ontwikkelingscyclus

Functionaliteit: Dit gaat over of de software doet wat het moet doen. Het is vooral belangrijk in de vereisten- en ontwerpfase.

Betrouwbaarheid: Dit heeft te maken met hoe goed de software werkt onder verschillende omstandigheden. Het is van belang tijdens de test- en onderhoudsfase.

Bruikbaarheid: Dit gaat over hoe gemakkelijk het is voor gebruikers om de software te begrijpen en te gebruiken. Het moet in alle fasen van de ontwikkelingscyclus worden overwogen.

Efficiëntie: Dit heeft betrekking op hoe goed de software presteert in termen van middelen zoals CPU, geheugen en schijfruimte. Het is vooral belangrijk in de implementatie- en testfase.

Veiligheid: Dit gaat over hoe goed de software is beschermd tegen bedreigingen. Het moet in alle fasen van de ontwikkelingscyclus worden overwogen.

Onderhoudbaarheid: Dit gaat over hoe gemakkelijk het is om de software te veranderen. Het is vooral belangrijk in de onderhoudsfase.

Overdraagbaarheid: Dit heeft betrekking op hoe gemakkelijk de software kan worden overgezet naar een andere omgeving. Het is vooral belangrijk in de implementatie- en onderhoudsfase.

Compatibiliteit: Dit gaat over hoe goed de software samenwerkt met andere systemen. Het is van belang tijdens de implementatie- en testfase.

4.3.2 Hoe elke Hoofdkenmerk zichtbaar wordt in de Software Ontwikkelingscyclus

In de volgende delen van dit verslag zullen we de acht kenmerken van ISO 25010, zoals hierboven beschreven, verder onderzoeken en toepassen op de verschillende fasen van de softwareontwikkelingscyclus. We zullen dit doen door specifieke voorbeelden te geven van hoe deze kenmerken zich manifesteren in ons project, de MVCLibraryApp.

Bijvoorbeeld, tijdens de eisenfase van ons project kijken we naar functionaliteit en bruikbaarheid. We stellen vragen als: "Wat moet onze app doen?" en "Hoe zorgen we ervoor dat gebruikers de app gemakkelijk kunnen gebruiken?" We stellen ook eisen op het gebied van veiligheid om te zorgen dat gebruikersdata veilig is.

Tijdens de ontwerpfase zullen we ons vooral richten op betrouwbaarheid en overdraagbaarheid. We willen ervoor zorgen dat ons ontwerp solide is, zodat de software niet crasht of faalt. We willen ook dat ons ontwerp flexibel is, zodat de software gemakkelijk kan worden aangepast of overgezet naar andere platforms.

Tijdens de implementatiefase kijken we naar efficiëntie en compatibiliteit. We willen dat onze code zo geschreven is dat de software efficiënt gebruik maakt van middelen. We willen ook dat onze software goed samenwerkt met andere systemen.

Tijdens de testfase controleren we alle bovengenoemde kenmerken, maar we richten ons specifiek op betrouwbaarheid en veiligheid om te zorgen dat de software onder alle omstandigheden goed werkt en veilig is.

Tenslotte, in de onderhoudsfase kijken we vooral naar onderhoudbaarheid. We willen ervoor zorgen dat de software gemakkelijk te veranderen en te verbeteren is naarmate de eisen van gebruikers veranderen.

Door elk van deze kenmerken in de juiste fase van de softwareontwikkelingscyclus te overwegen, kunnen we ervoor zorgen dat onze software voldoet aan de hoge standaard die ISO 25010 voorstelt.

5. ISO 25040

5.1 Begrip van ISO 25040

ISO 25040 is een belangrijk onderdeel van de ISO 25000 reeks, ook wel bekend als SQuaRE (System and Software Quality Requirements and Evaluation). Het specificeert de proces- en productkwaliteitseisen en biedt een evaluatieschema om de kwaliteit van softwareproducten te beoordelen.

5.2 ISO 25040 voor een Software Ontwikkelaar

Als software ontwikkelaars is het belangrijk dat wij ISO 25040 begrijpen en toepassen in ons werk. Het helpt ons om meetbare doelen te stellen voor productkwaliteit en biedt een kader voor het evalueren of deze doelen zijn bereikt.

5.3 De Onderdelen van ISO 25040

ISO 25040 bestaat uit drie hoofdonderdelen:

- **Evaluatieproces:** Dit beschrijft de stappen die moeten worden gevolgd om de kwaliteit van een softwareproduct te evalueren.
- **Evaluatiemodellen:** Deze bieden specifieke modellen en richtlijnen voor het meten van verschillende aspecten van softwarekwaliteit.
- **Kwaliteitsmetingen:** Dit onderdeel biedt richtlijnen voor het uitvoeren van metingen om de kwaliteit van een softwareproduct te beoordelen.

5.4 Het Gebruik van ISO 25040 in de Software Ontwikkelingscyclus

ISO 25040 kan op verschillende punten in de software ontwikkelingscyclus worden toegepast. Tijdens de planning en ontwerpfase kunnen we het gebruiken om kwaliteitsdoelen te stellen en de maatregelen die we zullen gebruiken om die doelen te meten te specificeren. Tijdens de ontwikkelingsfase kunnen we het gebruiken om de kwaliteit van onze code en onze processen te evalueren. Tijdens de testfase kunnen we het gebruiken om de effectiviteit van onze tests te beoordelen en tijdens de implementatiefase kunnen we het gebruiken om te beoordelen hoe goed onze software voldoet aan de eisen van de gebruiker.

6. Minimum Viable Product (MVP)

6.1 Begrip van MVP

Binnen de context van softwareontwikkeling is een Minimum Viable Product (MVP) een ontwikkelingsstrategie waarbij een nieuw product of een nieuwe functie met voldoende functies wordt geleverd om te voldoen aan de eerste gebruikers. Het eindproduct is vaak niet perfect, maar het heeft net genoeg functies om waarde te leveren en feedback van de gebruikers te verzamelen.

6.2 Voor- en Nadelen van een MVP

Het gebruik van een MVP-strategie heeft zowel voordelen als nadelen. De belangrijkste voordelen zijn snelheid en efficiëntie; door alleen te focussen op kernfuncties, kunnen we snel een werkend product leveren. Dit stelt ons in staat om snel feedback te verzamelen en iteraties te maken op ons product.

Echter, er zijn ook enkele nadelen verbonden aan een MVP. Ten eerste kan het zijn dat de kwaliteit van het product onder de MVP-strategie lijdt, aangezien sommige aspecten van de software mogelijk over het hoofd worden gezien in de haast om het product op de markt te brengen. Ten tweede kan het ook zijn dat sommige gebruikers teleurgesteld zijn als het product niet alle functies bevat die ze verwachten.

7. ISO 9000 en 9001

7.1 Begrip van ISO 9000 en 9001

ISO 9000 is een reeks standaarden voor kwaliteitsmanagement die door de Internationale Organisatie voor Standaardisatie (ISO) zijn vastgesteld. De reeks bestaat uit verschillende standaarden, maar ISO 9001 is de meest bekende en de enige in de reeks waarop organisaties gecertificeerd kunnen worden.

ISO 9001 specificeert eisen voor een kwaliteitsmanagementsysteem (QMS) dat een organisatie kan helpen bij het verbeteren van de algemene prestaties en het leveren van voldoening aan de klant. Het stelt organisaties in staat om te laten zien aan klanten dat ze een goed QMS hebben en dat ze zich houden aan de principes van kwaliteitsmanagement.

7.2 Quality Management Principles (QMP's) van ISO 9001

ISO 9001 is gebaseerd op zeven Quality Management Principles (QMP's). Deze zijn:

1. Klantgerichtheid
2. Leiderschap
3. Betrokkenheid van mensen
4. Procesbenadering
5. Verbetering
6. Besluitvorming op basis van bewijs
7. Relatiebeheer

Deze principes vormen de basis van de eisen van ISO 9001 en helpen organisaties om zich te richten op het bereiken van kwaliteit in hun processen en uitkomsten.

7.3 Voor- en Nadelen van ISO 9000 en 9001

Het toepassen van de ISO 9000 en 9001 reeks heeft verschillende voordelen. Het helpt bij het verbeteren van de kwaliteit en efficiëntie van de organisatie, verbetert de klanttevredenheid en verhoogt de concurrentiekracht op de markt. Bovendien kan de naleving van de standaarden de reputatie van de organisatie versterken en nieuwe zakelijke kansen bieden.

Er zijn echter ook enkele nadelen. Het implementeren en onderhouden van een QMS volgens ISO 9001 kan tijdrovend en kostbaar zijn. Daarnaast kunnen sommige organisaties het moeilijk vinden om de brede eisen van de standaard aan te passen aan hun specifieke situatie.

8. Functional en Non-Functional Requirements

8.1 Begrip van Functional Requirements

Functionele requirements beschrijven wat een systeem moet doen. Ze zijn gebaseerd op de functies en mogelijkheden die een systeem moet bieden aan de eindgebruiker. Dit omvat specifieke functionaliteiten die de software moet kunnen uitvoeren en de interacties die de gebruiker met het systeem zal hebben.

8.2 Begrip van Non-Functional Requirements

Non-Functionele Requirements daarentegen zijn de vereisten die de prestaties van het systeem of de operationele eigenschappen die het systeem moet hebben beschrijven. Dit omvat aspecten zoals betrouwbaarheid, bruikbaarheid, efficiëntie, onderhoudbaarheid, portabiliteit en veiligheid

8.3 Voorbeeldproject

In ons project, de MVCLibraryApp, hebben we een reeks functionele en non-functionele requirements geïdentificeerd die we zullen behandelen.

8.3.1 Vijf Functional Requirements en Testplan

1. **Zoeken en Reserveren van Materialen:** Gebruikers moeten in staat zijn om Items in de bibliotheek te zoeken en te reserveren. Dit omvat het zoeken op verschillende parameters zoals titel, auteur en locatie.
Testplan: Voer een reeks zoekopdrachten uit met verschillende parameters om te verifiëren dat de zoek- en reserveringsfunctie naar behoren werkt. Test ook het reserveringsproces om te bevestigen dat een gebruiker een item succesvol kan reserveren.
2. **Beheer van Leningen en Reserveringen:** Gebruikers moeten in staat zijn om hun eigen leningen en reserveringen te beheren.
Testplan: Test of gebruikers in staat zijn om leningen en reserveringen te bekijken, te verlengen, te annuleren en te beheren zoals verwacht.
3. **Beheer van Bibliotheekgegevens:** Personeel en beheerders moeten toegang hebben tot alle gegevens van de bibliotheek, inclusief uitleengegegevens en voorraadbeheer.
Testplan: Zorg ervoor dat de relevante beheertools beschikbaar zijn en naar behoren functioneren voor personeel en beheerders.
4. **Verwerking van Betalingen:** De applicatie moet in staat zijn om betalingen te verwerken voor eventuele kosten die in rekening worden gebracht voor abonnementen, uitgeleende materialen, reserveringskosten, boetes, enz.
Testplan: Voer transacties uit om te verifiëren of betalingen correct worden verwerkt en weergegeven in het systeem.

5. **Wijzigingen in Abonnementen:** De applicatie moet in staat zijn om eventuele wijzigingen in abonnementen gemakkelijk door te voeren.
Testplan: Voer verschillende scenario's uit waarbij abonnementsgegevens worden gewijzigd om te bevestigen dat deze wijzigingen correct worden doorgevoerd en weergegeven.

8.3.2 Vijf Non-Functional Requirements en Testplan

1. **Prestatie:** Het systeem moet in staat zijn om 1000 gelijktijdige gebruikers te ondersteunen zonder afbreuk te doen aan de snelheid van de respons.

Prestatietests: Voer een stresstest uit met 1000 gelijktijdige virtuele gebruikers en meet de reactietijd van het systeem.

2. **Beveiliging:** Het systeem moet alle gebruikersgegevens beschermen met encryptie tijdens opslag en verzending. Het moet voldoen aan de standaard voor databeveiliging, zoals de GDPR.

Beveiligingstests: Voer tests uit om de encryptie van gegevens tijdens opslag en verzending te controleren. Doe ook een penetratietest om te zien of er beveiligingslekken zijn.

3. **Betrouwbaarheid:** Het systeem moet een uptime van 99,9% hebben, wat betekent dat het systeem maximaal 8 uur per jaar offline kan zijn.

Betrouwbaarheidstests: Monitor het systeem gedurende een jaar om de uptime te meten. Controleer ook de back-up- en herstelprocedures van het systeem.

4. **Gebruiksvriendelijkheid:** Het systeem moet intuïtief en gemakkelijk te gebruiken zijn, zelfs voor nieuwe gebruikers. Het moet niet meer dan 10 minuten duren voor een nieuwe gebruiker om te leren hoe het systeem te gebruiken.

Gebruiksvriendelijkheidstests: Voer gebruikstests uit met echte gebruikers, met name nieuwe gebruikers, om de gebruiksvriendelijkheid van het systeem te evalueren.

5. **Schaalbaarheid:** Het systeem moet gemakkelijk kunnen worden opgeschaald om het groeiende aantal gebruikers te ondersteunen, zonder dat dit de prestaties beïnvloedt.

Schaalbaarheidstests: Voer loadtests uit op het systeem door geleidelijk het aantal gebruikers te verhogen en te observeren hoe het systeem zich gedraagt.

9. Continuous Integration and Continuous Delivery (CI/CD)

9.1 Begrip van CI/CD en CI/CD Pipeline

CI/CD staat voor Continuous Integration en Continuous Delivery. Dit is een methode om softwareontwikkeling te versnellen en te stroomlijnen door regelmatig code-updates naar het hoofdproject te pushen.

- **Continuous Integration (CI):** Dit is het proces waarbij ontwikkelaars regelmatig (vaak dagelijks) nieuwe code in een gemeenschappelijke repository integreren. Elke integratie wordt vervolgens automatisch getest om vroege detectie van fouten of problemen mogelijk te maken.
- **Continuous Delivery (CD):** Dit is het proces waarbij softwareveranderingen automatisch worden voorbereid voor een release naar productie. Het doel is om een systeem te hebben dat klaar is voor een release op elk moment.

Een CI/CD Pipeline is een reeks stappen die je code doorloopt vanaf het moment dat een ontwikkelaar een wijziging aanbrengt tot het moment dat deze in productie gaat. Het omvat meestal fasen zoals het bouwen van de code (compilatie, linter, unit tests), het testen van de code (integratie tests, systeemtests) en het uitrollen van de code naar productie (staging, productie).

9.2 MVCLibraryApp CI/CD Pipeline

1. **Code** - Ontwikkelaars werken aan de MVC Library App op hun lokale machines, toevoegen nieuwe functies, verbeteren de UI, voeren bugfixes uit, enz. Ze gebruiken Git voor versiebeheer om hun wijzigingen bij te houden.
2. **Commit** - Ontwikkelaars maken regelmatig commits en pushen die naar een gecentraliseerde Git-repository op een platform zoals GitHub of Bitbucket.
3. **Build** - Wanneer nieuwe commits worden gedetecteerd in de repository, start de CI/CD-tool (zoals Jenkins of CircleCI) een nieuwe build. Deze bouwphase omvat de compilatie van de code, de uitvoering van linters (bijvoorbeeld StyleCop voor C#) en het uitvoeren van unit tests (bijvoorbeeld met behulp van NUnit of xUnit.net).
4. **Test** - Na een succesvolle bouwphase voert de CI/CD-tool integratietests uit op de code. Dit kan bijvoorbeeld het testen van de interacties tussen de controllers en views in je MVC-applicatie, of de interacties met de database.
5. **Deploy to Staging** - Als de tests succesvol zijn, wordt de code uitgerold naar een stagingomgeving. Deze omgeving moet zo veel mogelijk op de productieomgeving lijken. Hier kan je team handmatige tests uitvoeren en de functionaliteit van de nieuwe features bevestigen in een veilige omgeving.
6. **User Acceptance Testing (UAT)** - In deze fase kan de klant of eindgebruiker de toepassing testen en feedback geven. Deze fase is belangrijk om ervoor te zorgen dat de applicatie voldoet aan de behoeften en verwachtingen van de eindgebruiker.
7. **Deploy to Production** - Na succesvolle UAT wordt de code uitgerold naar de productieomgeving. Dit is de "live" versie van de applicatie die door eindgebruikers zal worden gebruikt.
8. **Monitor** - Na de release wordt de applicatie in productie gemonitord om de prestaties te controleren en eventuele problemen snel te identificeren.

Elke stap in deze pijplijn moet zo geautomatiseerd mogelijk zijn om menselijke fouten te minimaliseren en de snelheid van levering te verhogen. CI/CD is een krachtige methode om ervoor te zorgen dat je team regelmatig kwaliteitscode kan leveren.

10. Testen

10.1 Begrip en Soorten van Testen

Testen in softwareontwikkeling is het proces van het evalueren van een systeem of systeemcomponent tijdens of na de ontwikkeling ervan. Het doel is om te bepalen of het voldoet aan de specificaties of om verschillende eigenschappen te ontdekken. Er zijn verschillende soorten tests, zoals:

- Unit-tests: Testen van afzonderlijke componenten van een systeem, zoals functies of methoden, om te verifiëren dat ze correct werken.
- Integratie-tests: Testen van de interacties tussen verschillende componenten van een systeem om te verifiëren dat ze correct samenwerken.
- Systeemtests: Testen van een volledig systeem om te verifiëren dat het voldoet aan de specificaties.
- Acceptatietests: Testen om te verifiëren of het systeem voldoet aan de eisen van de gebruiker of klant.
- Loadtests: Testen van de systeemprestaties onder belasting, zoals een groot aantal gebruikers of grote hoeveelheden gegevens.

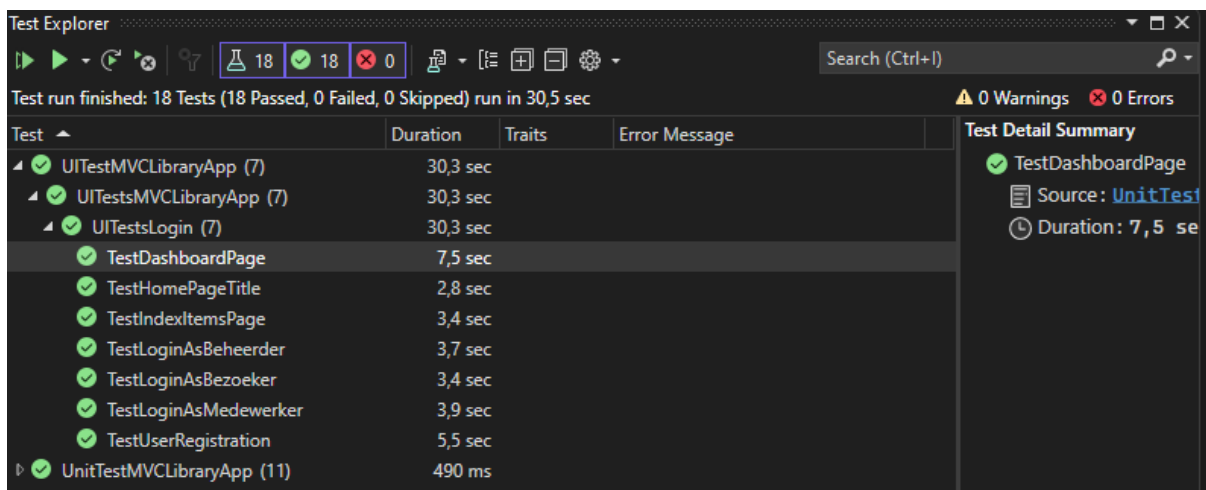
10.2 Test Uitwerkingen op MVCLibraryApp

Alle tests zijn terug te vinden in de Github Repository:

<https://github.com/SchaapKantierLibrary/MVCLibraryApp.git>.

10.2.1 UI-test

Voor de UI tests is gebruik gemaakt van een combinatie van Nunit en Selenium, gecombineerd met de WebDriverManager packages. terug te vinden in de Github onder de solution MVCLibraryApp, project UITestMVCLibraryApp.



The screenshot shows the Test Explorer window in Visual Studio. At the top, it indicates 'Test run finished: 18 Tests (18 Passed, 0 Failed, 0 Skipped) run in 30,5 sec'. Below this, a table lists the tests and their durations. On the right, the 'Test Detail Summary' for 'TestDashboardPage' is shown, indicating it passed and took 7,5 seconds.

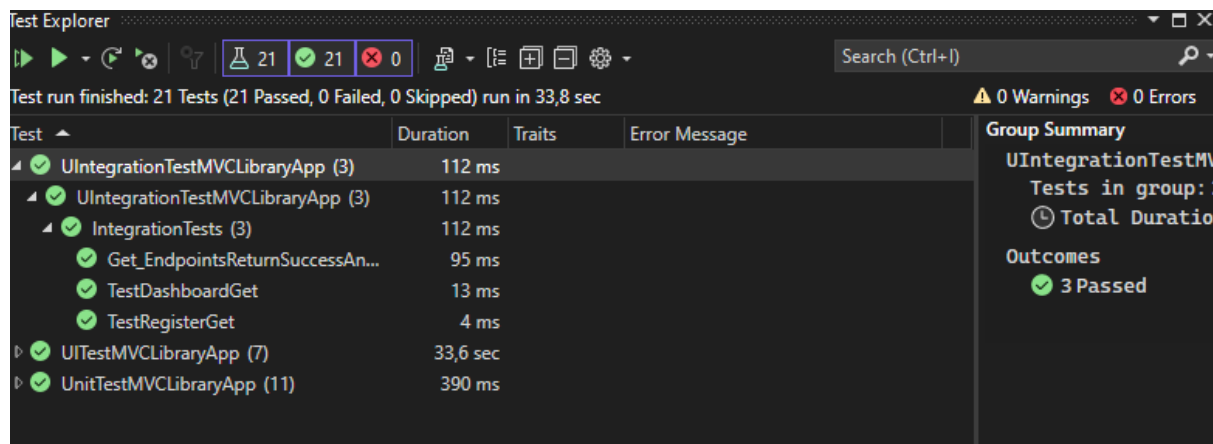
Test	Duration	Traits	Error Message
UITestMVCLibraryApp (7)	30,3 sec		
UITestsMVCLibraryApp (7)	30,3 sec		
UITestsLogin (7)	30,3 sec		
TestDashboardPage	7,5 sec		
TestHomePageTitle	2,8 sec		
TestIndexItemsPage	3,4 sec		
TestLoginAsBeheerder	3,7 sec		
TestLoginAsBezoeker	3,4 sec		
TestLoginAsMedewerker	3,9 sec		
TestUserRegistration	5,5 sec		
UnitTestMVCLibraryApp (11)	490 ms		

Test Detail Summary

- TestDashboardPage
- Source: [UnitTest](#)
- Duration: 7,5 se

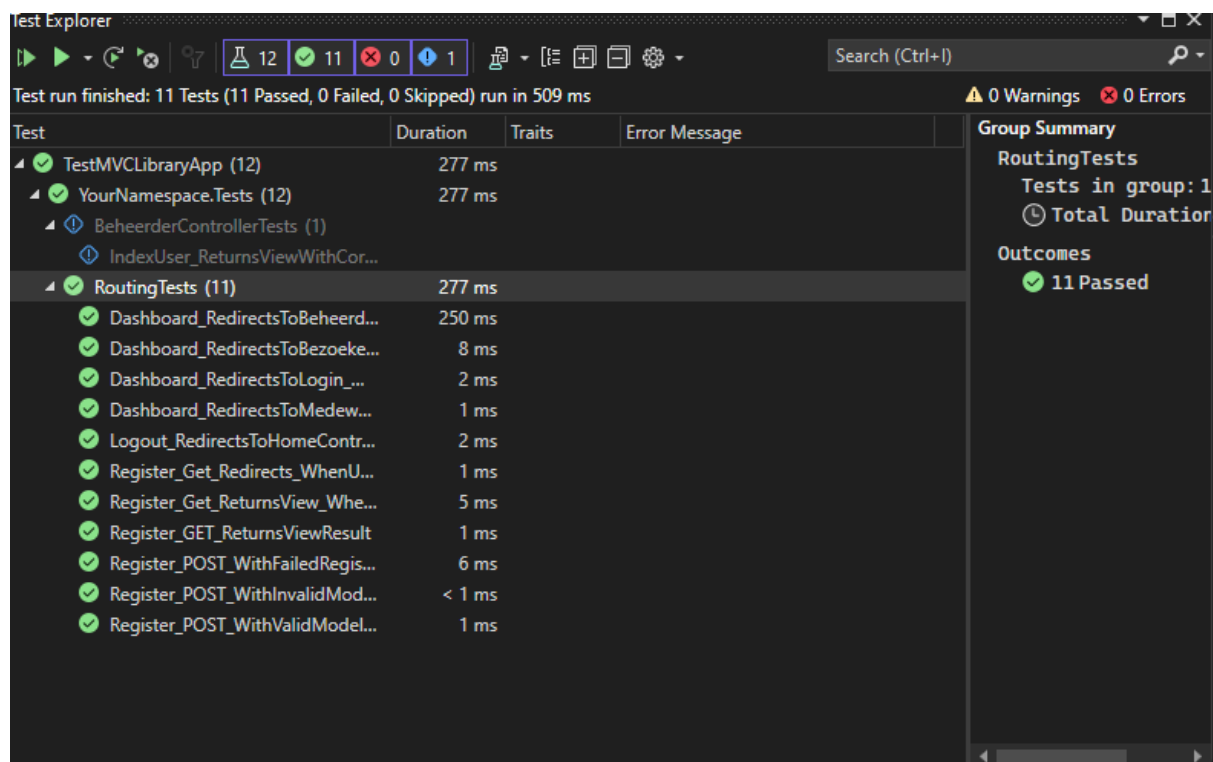
10.2.2 Integratie-test

Gebruik gemaakt van Nunit voor de Integratietesten, terug te vinden in de Git repository, onder de Solution MVCLibraryApp, project UIntegrationTestMVCLibraryApp.



10.2.3 Unit-tests

Voor De Unit tests is gebruik gemaakt van de Nunit Package. Terug te vinden in de Github in het project "UnitTestMVCLibraryApp" in de MVCLibraryApp solution.



10.3 Complementaire Natuur van Testen

Testen binnen softwareontwikkeling hebben een complementaire aard. Verschillende soorten tests richten zich op verschillende aspecten van de software en vullen elkaar aan om een grondige evaluatie te bieden. Hierdoor kunnen potentiële problemen worden geïdentificeerd en opgelost voordat de software in gebruik wordt genomen.

Unit-tests zijn gericht op het testen van individuele componenten om ervoor te zorgen dat ze correct werken. Ze kunnen problemen in de code identificeren, zoals logische fouten of onverwachte resultaten.

Integratietests richten zich op het testen van de interacties tussen verschillende componenten om te verifiëren dat ze correct samenwerken. Dit is belangrijk om ervoor te zorgen dat de verschillende delen van het systeem goed geïntegreerd zijn en goed functioneren als een geheel.

Systeemtests testen het volledige systeem om te verifiëren dat het voldoet aan de specificaties en de verwachtingen van de gebruiker. Dit omvat het testen van zowel de functionaliteit als de niet-functionele aspecten van de software.

Acceptatietests zijn gericht op het testen van het systeem om te verifiëren of het voldoet aan de eisen van de gebruiker of klant. Dit omvat het testen van de software in een realistische gebruikersomgeving om ervoor te zorgen dat het voldoet aan de verwachtingen van de gebruiker.

Loadtests zijn gericht op het testen van de systeemprestaties onder belasting. Dit omvat het testen van de prestaties van het systeem met een groot aantal gebruikers of grote hoeveelheden gegevens. Deze tests zijn belangrijk om ervoor te zorgen dat het systeem robuust en schaalbaar is en goed presteert onder verschillende belastingcondities.

Door het uitvoeren van een combinatie van deze tests kunnen ontwikkelaars een grondige evaluatie van de software uitvoeren en potentiële problemen identificeren. Ze bieden een complementaire benadering van het testen van verschillende aspecten van de software en dragen bij aan het bereiken van een hoog niveau van kwaliteit en betrouwbaarheid.

10.4 Implementatie van Testen in een Organisatie

Om testen succesvol te implementeren in een organisatie, is het belangrijk om een gestructureerde aanpak te volgen en betrokkenheid op verschillende niveaus te waarborgen. Hier zijn enkele essentiële aspecten die moeten worden overwogen:

1. **Teststrategie en beleid:** Definieer een passende teststrategie en stel een duidelijk testbeleid op om richtlijnen en standaarden voor testen vast te leggen.
2. **Testplanning en resources:** Ontwikkel een gedetailleerd testplan dat de testactiviteiten, planning, benodigde middelen en verantwoordelijkheden omvat.
3. **Testautomatisering:** Maak gebruik van testautomatiseringstools en frameworks om de efficiëntie en herbruikbaarheid van tests te verbeteren.
4. **Testomgeving:** Zorg voor een testomgeving die zo nauwkeurig mogelijk overeenkomt met de productieomgeving om realistische testresultaten te verkrijgen.
5. **Testdata:** Verzeker de beschikbaarheid van voldoende en representatieve testdata, terwijl de vertrouwelijkheid en privacy van gegevens worden gewaarborgd.
6. **Testuitvoering en rapportage:** Voer tests uit volgens het testplan, documenteer nauwkeurig de resultaten en communiceer over de voortgang en eventuele risico's.
7. **Testprocesverbetering:** Monitor en evalueer het testproces voortdurend om inefficiënties en verbeterpunten te identificeren.
8. **Samenwerking en communicatie:** Stimuleer samenwerking tussen ontwikkelings- en testteams en communiceer duidelijk over testresultaten en bevindingen.
9. **Training en kennisdeling:** Zorg voor de juiste training en kennisdeling op het gebied van testen om de vaardigheden van testers en ontwikkelaars te vergroten.
10. **Kwaliteitscultuur:** Creëer een cultuur waarin testen als een integraal onderdeel van het ontwikkelproces wordt gezien en waarin de waarde van testen wordt erkend.

Door deze aspecten in acht te nemen en de implementatie van testen serieus te nemen, kan een organisatie een solide basis leggen voor het leveren van hoogwaardige softwareproducten

10.5 Belang van Testen

Het belang van testen binnen softwareontwikkeling kan niet worden overschat. Testen speelt een cruciale rol bij het waarborgen van de kwaliteit, betrouwbaarheid en bruikbaarheid van software. Hieronder volgen enkele belangrijke redenen waarom testen van essentieel belang is:

- **Vroegtijdige probleemidentificatie:** Door middel van testen kunnen ontwikkelaars potentiële problemen en fouten in een vroeg stadium identificeren. Het vroegtijdig opsporen van problemen stelt hen in staat om deze efficiënt en kosteneffectief op te lossen.
- **Kwaliteitsborging:** Testen helpt bij het waarborgen van de kwaliteit van de software. Door verschillende soorten tests uit te voeren, kunnen ontwikkelaars de functionaliteit, prestaties en veiligheid van de software valideren en verifiëren.
- **Klanttevredenheid:** Goed geteste software leidt tot hogere klanttevredenheid. Het minimaliseert de kans op bugs, crashes en onverwacht gedrag, waardoor gebruikers een positieve ervaring hebben en vertrouwen in de software krijgen.
- **Risicobeheer:** Testen helpt bij het identificeren en beheren van risico's die verband houden met de software. Door risico's in kaart te brengen en te beoordelen, kunnen maatregelen worden genomen om de impact ervan te minimaliseren.
- **Betrouwbaarheid en stabiliteit:** Grondig testen zorgt voor betrouwbaarheid en stabiliteit van de software. Dit is vooral belangrijk voor software die kritieke taken uitvoert of wordt gebruikt in gevoelige omgevingen.
- **Kostenbesparing:** Hoewel testen tijd en middelen vereist, kan het uiteindelijk kosten besparen. Door problemen vroegtijdig aan te pakken, worden dure herstelacties en reputatieschade voorkomen die kunnen ontstaan door het uitrollen van gebrekkige software.
- **Continu verbeteren:** Testen draagt bij aan een continu verbeteringsproces binnen een organisatie. Door feedback van tests te analyseren, kunnen ontwikkelaars inzichten verkrijgen die kunnen worden gebruikt om de software en het ontwikkelproces te optimaliseren.

Al met al is testen een integraal onderdeel van softwareontwikkeling. Het biedt ontwikkelaars de nodige zekerheid dat de software goed functioneert, voldoet aan de vereisten en tegemoetkomt aan de verwachtingen van de gebruikers. Het investeren in testen is een waardevolle inspanning die leidt tot betere software en tevreden klanten.

11. Conclusie

In dit verslag hebben we verschillende kernconcepten binnen softwareontwikkeling besproken. We begonnen met het definiëren van kwaliteit en software, en hoe kwaliteit in software verschilt van algemene kwaliteit. Vervolgens hebben we gekeken naar de ISO 25010 en ISO 25040 normen, die beide helpen bij het meten en verbeteren van de kwaliteit van softwareproducten.

We hebben ook het concept van een Minimum Viable Product (MVP) besproken, waarbij we een product ontwikkelen met voldoende functies om aan de eerste gebruikers te voldoen en feedback te verzamelen. Daarnaast hebben we gekeken naar de ISO 9000 en 9001 normen voor kwaliteitsmanagement en de principes die hieraan ten grondslag liggen.

Verder hebben we het verschil tussen functionele en niet-functionele eisen besproken en hebben we voorbeelden gegeven van functionele en niet-functionele requirements voor ons voorbeeldproject, de MVCLibraryApp. We hebben ook de Continuous Integration and Continuous Delivery (CI/CD) methode besproken, die helpt bij het versnellen en stroomlijnen van softwareontwikkeling, en hebben de verschillende soorten tests behandeld, zoals unit-tests, integratietests en systeemtests.

Al deze concepten en methoden zijn essentieel om de kwaliteit van softwareproducten te waarborgen en effectieve softwareontwikkeling te bevorderen. Door ze toe te passen in onze softwareprojecten, kunnen we ervoor zorgen dat onze software goed werkt, betrouwbaar is, gemakkelijk te gebruiken is en voldoet aan de eisen van de gebruikers.

12. Referenties

[1] ISO 25010:2011 - Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.

[2] ISO 25040:2011 - Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Evaluation process.

[3] ISO 9000:2015 - Quality management systems -- Fundamentals and vocabulary.

[4] ISO 9001:2015 - Quality management systems -- Requirements.

[5] Martin Fowler. (2014). Continuous Integration.
<https://martinfowler.com/articles/continuousIntegration.html>

[6] ISTQB Glossary of Testing Terms. (2018). International Software Testing Qualifications Board (ISTQB).

13. Reflectie

13.1 Reflectieverslag (Quinten Schaap):

Ik vond het interessant om me te verdiepen in de ISO-normen en CI/CD. Het begrijpen van de complementaire aard van testen was ook waardevol. Samenwerken met Robin was prettig en we hebben een mooi resultaat geleverd.

13.1.1 Feedbackverslag (Quinten Schaap voor Robin Kantier):

Robin leverde een waardevolle bijdrage aan het verslag. Het Onderzoek wat hij heeft gepleegd en gedocumenteerd was substantieel, Hij heeft begrip van softwarekwaliteit en softwareontwikkeling. Door de structuur die Robin zijn manier van werken mij gaf, hebben wij samen veel kunnen doen. Ik ben tevreden met onze samenwerking.

13.2 Reflectieverslag (Robin Kantier):

Ik vond het een interessant onderwerp om te verwerken in een verslag. Het onderzoeken en uitleggen van de ISO-normen en CI/CD gaven me een dieper inzicht in softwarekwaliteit, en hoe ik dit kan implementeren tijdens de development van mijn projecten. Ik heb veel geleerd over de complementaire aard van het testen, en de implementatie hiervan op mijn ontwikkelomgeving. De samenwerking met Quinten verliep soepel en ik ben tevreden met het eindresultaat.

13.2.1 Feedbackverslag (Robin Kantier voor Quinten Schaap):

Quinten leverde een waardevolle bijdrage aan het verslag. Zijn onderzoek en uitleg waren grondig en duidelijk. Hij toonde interesse in het onderwerp en was betrokken bij het proces. Quinten was betrouwbaar en ik waardeerde onze samenwerking.