Automated Software Validation

Hogeschool van Amsterdam

Door

Sirée Koolen-Wijkstra	500544674
Tamara Goode	500606403

Discover new Requirements **2**
Initiële proces 2
Elicitation Technique 3
Questionnaires 4
Vergelijking technieken 4
Requirements 5
Baseline assessment **6**
Relevante code 6
Maintainability 8
Tracker solution 8
Consumer solution 9
WebApi solution 10
SkyWebApp solution 11
Coverage 12
Tracker solution 12
Consumer solution 13
WebApi solution 14
SkyWebApp solution 15
Cyclomatic complexity 16
Tracker solution 16
Consumer solution 16
WebApi solution 17
SkyWebApp solution 18
Assessment with Sonarqube including Unit Tests **19**
Unit Test **21**
First en Right BICEP 22
WebApi AircraftTrackingInfoControllerTests 23
Entity AircraftTrackInfoTests, AircraftArrivalInfoTests en AircraftDepartureInfoTests 24
Consumer ProgramTests 25
Tracker ProgramTests 26
SkyWebApTests 27
Pipeline in Azure DevOps **28**
Reflectie **31**
Appendix A **32**

Discover new Requirements

Initiële proces

Aanvankelijk hebben we de technieken ‘het schrijven van use cases’ en ‘het houden van een open interview’ met onze opdrachtgever gebruikt. Dit heeft geresulteerd in een twaalftal requirements gebaseerd op quality attributes, die later gespecificeerd worden.

De reden waarom we voor de techniek ‘het houden van een open interview’ hebben gekozen is, omdat we inschatten dat ons beider level als elicitor low is voor wat betreft training in elicitation techniques, experience met elicitation en elicitation techniques. We hebben beide immers veel elicitation technieken toegepast in de verschillende projecten van school. Daarnaast is Sirée werkzaam als functioneel beheerder, wat betekent dat ze regelmatig wensen van gebruikers vertaalt naar wijzigingsverzoeken voor de leverancier. Tamara is werkzaam als software architect, ze voert vanuit die rol regelmatig gesprekken met stakeholders over processen en wensen.

Het is de wens van de opdrachtgever (het softwarebedrijf waarvoor we de functionaliteit bouwen) om geen contact te zoeken met klanten, een andere stakeholder. De laatste stakeholder, de investeerder, zou ons geen informatie kunnen geven over de gewenste functionaliteit, zij zijn om die reden niet benaderd.

De informant is individueel. Omdat hij individueel is, is er ook een hoog niveau van consensus. Hij is knowledgeable over de software die zijn bedrijf aanbiedt, maar geen expert. Hij is immers leidinggevende en niet uitvoerende in het bedrijf. Hij kan op een medium niveau articuleren wat de eisen zijn. Het was de bedoeling dat er meerdere open interviews gehouden zouden worden, maar de availability van de opdrachtgever is weggezakt naar low. De voorbereidingen voor een pilot bij een klant duren langer dan aanvankelijk gepland, waardoor er weinig tijd is voor de nice-to-have functionaliteit die wij gaan bouwen. Availability of time is low. Wat het houden van het interview vergemakkelijkte is dat Tamara werkt voor de opdrachtgever. De location is dan ook zeer near.

We hebben gekozen voor het schrijven van use cases, omdat Tamara erg bekend is met het domein (familiarity is high). Zij werkt al immers enige tijd voor het bedrijf. De familiarity van Siree met het domein is zero. Omdat Tamara goed bekend is met het domein en omdat ze het bedrijf ook goed kent, konden we goed inschatten hoe de MVP’s zoals beschreven in de use cases kunnen verlopen. De problem definiteness is high. De gewenste functionaliteit is duidelijk. De type of information van het Problem domain is tactical van aard, het gaat over het maken van de procedures. Tamara wist ook te vertellen dat de level of available information lower is. Er is namelijk wel veel algemene kennis over de toegepaste architectuur binnen het bedrijf, maar het bedrijf zelf heeft weinig tot geen beschrijvingen van z’n eigen infrastructuur en configuratie. Door middel van een use case konden we bepalen aan welke requirements voldaan moet worden voor een succesvolle MVP.

De factor process speelt een rol in het aantal geschreven use cases. Vanwege de high constraint aan project time hebben we drie use cases geschreven die we verwachten te realiseren. Onze process time is middle, wat wil zeggen dat de key requirements vooral heel helder moeten komen.

De requirements die uit het interview zijn gekomen, zijn in het interview gelijk gevalideerd. Er is gelijk mondeling feedback gegeven door de opdrachtgever. De requirements die uit de use case naar voren zijn gekomen, zijn niet gevalideerd door de opdrachtgever wegens zijn beperkte beschikbaarheid.

Elicitation Technique

Aantal	**Techniek**
14	Questionnaires
13	Task observation
12	Repetory grid, Nominal group T, Participant oberv., Scenario/Use cases.

Table 5
Adequacy of elicitation techniques for contextual characteristics.

Factor	Attributes	Values	Open-ended interv.	Structured interv.	Task observation	Card sorting/ ladd.	Questionnaires	Protocol analysis	Repertory grid	Brain-storming	Nominal group T.	Delphi technique	Participant observ.	Proto-typing	Focus group	JAD workshop	Scenarios/ U. cases
			10	11	13	11	14	8	12	9	12	11	12	11	9	6	12
Elicitor	Training in elicitation techniques	High	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
		Low	–	√	√	–	√	–	–	√	√	–	√	–	–	–	–
		Zero	–	–	–	×	–	×	×	–	×	×	–	–	×	×	–
	Elicitation experience	High	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
		Medium	√	√	√	√	√	–	√	–	√	√	√	√	√	–	√
		Low	–	–	√	–	–	×	–	–	√	√	√	–	–	×	–
	Experience with elicitation techniques	High	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
		Low	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
		Zero	–	–	√	–	√	–	–	–	√	–	–	–	–	×	–
	Familiarity with domain	High	√	√	–	√	√	–	√	√	–	√	–	√	√	√	–
		Low	√	√	–	√	√	√	√	√	–	√	√	–	√	√	–
		Zero	√	–	√	–	–	√	–	√	–	–	√	×	×	×	–
Informant	People per session	Individual	√	√	√	√	√	√	√	×	×	×	√	√	×	×	√
		Group	–	–	√	–	√	–	√	–	√	√	√	–	√	–	–
		Mass	×	×	–	×	√	×	√	√	√	√	√	–	√	√	–
	Consensus among informants	High	√	√	–	√	√	√	√	–	√	–	–	√	√	–	√
		Low	×	×	–	×	√	×	√	–	√	√	–	√	√	–	–
	Informant interest	High	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
		Low	–	–	√	√	√	–	√	–	√	√	–	√	√	–	√
		Zero	×	–	√	–	√	×	√	×	√	√	×	√	–	×	–
	Expertise	Expert	√	√	–	√	√	√	√	√	√	√	√	√	√	√	√
		Knowledgeable	√	√	√	√	√	–	√	√	√	√	√	√	√	√	√
		Novice	–	–	√	√	√	–	√	√	√	–	√	√	√	√	√
	Articulability	High	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
		Medium	√	√	√	√	√	–	√	√	√	√	√	√	√	√	√
		Low	–	–	√	√	√	×	√	×	×	√	–	√	×	×	–
	Availability of time	High	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
		Low	–	–	√	–	√	×	√	×	×	√	×	–	×	×	–
	Location/ accessibility	Far	–	–	–	–	√	×	√	×	×	√	×	×	×	–	×
		Near	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
Problem domain	Type of information to be elicited	Strategic	√	–	–	×	√	–	×	√	√	√	√	–	√	√	–
		Tactical	√	√	√	×	√	√	×	√	√	√	√	√	√	√	√
		Basic	–	√	–	√	√	–	√	×	×	–	–	√	–	×	–
	Level of available information	Upper	√	√	√	×	√	√	×	–	–	√	√	√	√	√	√
		Lower	√	√	√	√	√	×	√	√	√	√	√	√	×	√	√
		Zero	√	×	√	×	×	×	×	√	√	×	√	×	×	√	√
	Problem definedness	High	–	√	√	√	√	√	√	√	√	√	√	√	√	×	√
		Low	√	–	√	×	√	×	×	√	√	√	–	√	√	√	–
Process	Project time constraint	High	–	–	×	√	√	×	√	×	×	×	×	×	×	×	√
		Medium	√	√	–	√	√	–	√	–	–	–	–	–	–	–	√
		Low	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
	Process time	Start	√	–	√	√	–	×	–	√	√	–	√	–	√	√	–
		Middle	√	√	–	√	√	√	√	–	√	√	√	√	√	×	√
		End	√	√	×	×	√	–	–	×	×	–	×	–	–	×	–

Questionnaires

In Appendix A bespreken we de questionnaires.

Vergelijking technieken

Onze zelfgekozen criteria voor requirements zijn voornamelijk gericht op suitability en performance. De reden hiervoor is dat in het interview met onze informant aangegeven is dat voor Sky Reckon de klantbeleving heel belangrijk is. Sky Reckon biedt een dienst die al bestaat en aangeboden wordt door een concurrent. Ze zitten nog in de fase waarin ze zich proberen te positioneren op de markt. Sky Reckon wil zich onderscheiden door betere performance en vriendelijkere user interface.
Op basis van onze eigen kennis van het domein hebben we in de use cases bedacht dat bij uitbreiding van het klantenbestand de uitbreiding van capaciteit een factor is waar rekening mee moet worden gehouden. Op basis van onze kennis hebben we ook bedacht dat de data geen persoonsgegevens bevat. Sky Reckon heeft op dit moment geen security requirements. Ze beroepen zich op SLA-afspraken met gelicenseerde derden waarbij ze serverruimte afnemen. Om toch in ieder geval inzicht te verkrijgen of er verdere maatregelen moeten worden genomen, hebben we als requirement dat er logging moet plaatsvinden.

De gestructureerde enquête heeft een aantal voordelen boven het schrijven van use cases en het houden van een open interview. Het heeft het voordeel dat we van te voren precies kunnen verzinnen welke informatie we moeten uitvragen. In een open interview wordt dit veel meer geleid door de informant, waardoor je meerdere sessies eigenlijk zou moeten organiseren. In onze situatie zijn de opvolgende sessies voor het houden van open interviews een probleem geworden.
Een ander voordeel van enquêteren is dat we niet zelf alles hoeven te verzinnen. Er bestaan zogeheten tactics[1] om de juiste architectuur uit te vragen. De kwaliteit van de vragen zijn daardoor van betere kwaliteit en vragen beter door.
Een nadeel van de enquêtes is wel dat het staat of valt met het kennisniveau van de informant. Omdat die bij ons maar beperkt beschikbaar is, is dat een probleem. Als hij het antwoord niet weet, dan hebben we geen antwoord op onze vragen. Als alleenstaand instrument zouden we het niet snel kiezen.

[1] Humberto Cervantes, Rick Kazman, April 28, 2016, *Designing Software Architectures: A Practical Approach*

Requirements

1. Data needs to be retrieved and matched with assets owned by user with 100% accuracy.
2. Data requests/updates should be handled in 3 seconds maximum.
3. Presentation of data should happen in 2 seconds maximum.
4. 50 data request send at the same time should be handled without noticeable time delay.
5. A 100 users should be able to use the functionality at the same time without noticeable delays.
6. With no more than 5 clicks, a user can request all data pertaining to his aircraft.
7. Input needs to be validated, and an error message is shown when wrong data is entered.
8. Downtime of software is at most 2 hours every time a docker is exchanged[2].

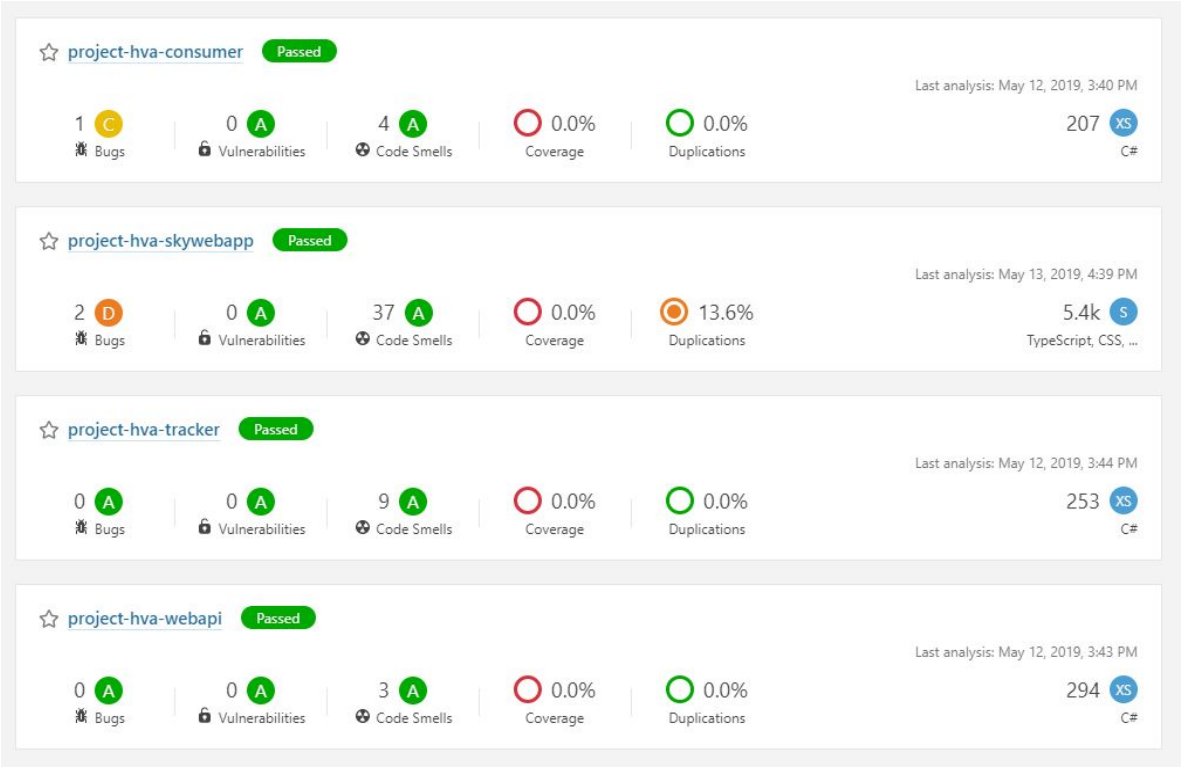
[2] Requirements zijn in het Engels omdat deze zijn vastgesteld samen met de opdrachtgever en hij communiceert in Engels.

Baseline assessment

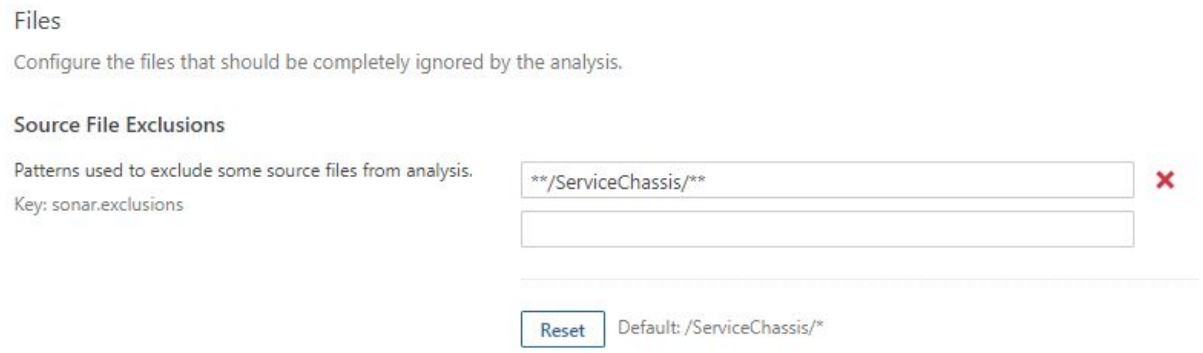
We hebben Sonarqube ingeregeld op de Tracker solution, de Consumer solution, de Web Api solution en de SkyWebApp solution van ons project. De Tracker solution haalt gegevens op uit een externe URL en vormt deze json in een nieuw jasje. Vervolgens sturen we deze informatie door naar de backend, Kafka. De Consumer solution is verantwoordelijk voor de communicatie tussen Kafka en Redis, waarbij Redis de querykant is die de Web Api solution gebruikt. De front end maakt connectie met de Web Api solution, waarbij de Web Api solution alle REST calls afhandelt. Alle back end solutions hebben middels het Entity project toegang tot de gedeelde classes, dit is geen aparte solution. Entity is onderdeel van elke solution. De SkyWebbApp solution verzorgt de front end.

Relevante code

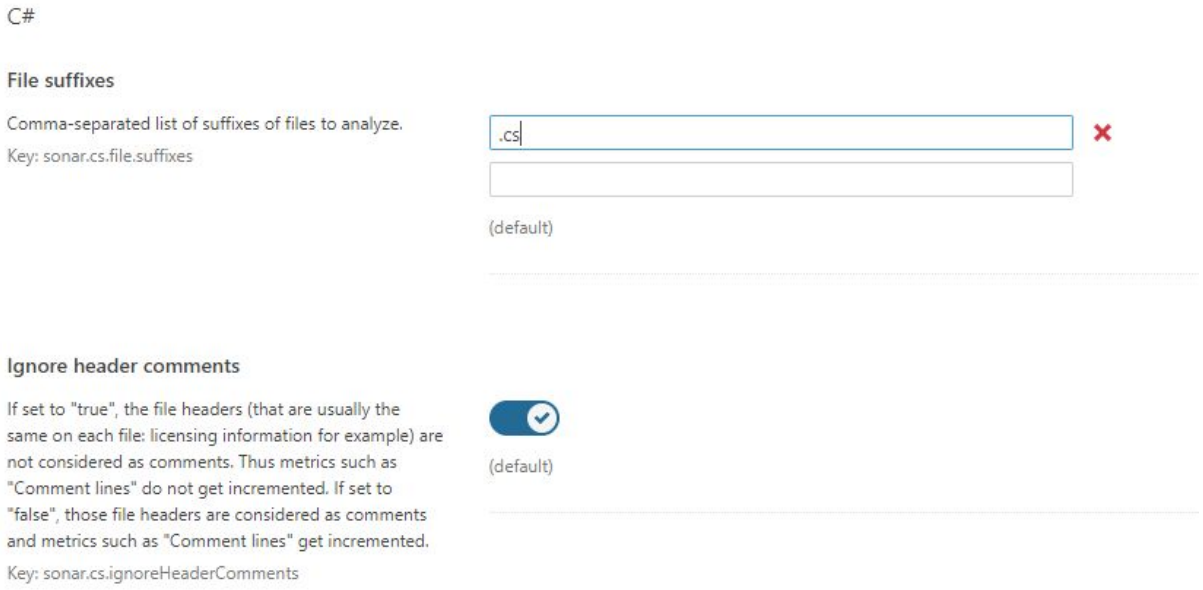
We hebben verschillende maatregelen genomen om alleen de relevante code te analyseren. Relevante code is in dit geval alleen de code die we zelf hebben geschreven voor het project. We hebben Sonarqube zo ingesteld dat alleen bovengenoemde solutions worden geanalyseerd. Sonarqube kan helaas maar één solution tegelijkertijd analyseren en dus hebben we voor iedere solution apart een sonarqube project moeten opzetten.



Onze code maakt gebruik van reeds bestaande, niet door ons gemaakte, code, genaamd ServiceChassis. Deze code sluiten we uit.



Een algemene maatregel is het negeren van alle files zonder een .cs extensie.

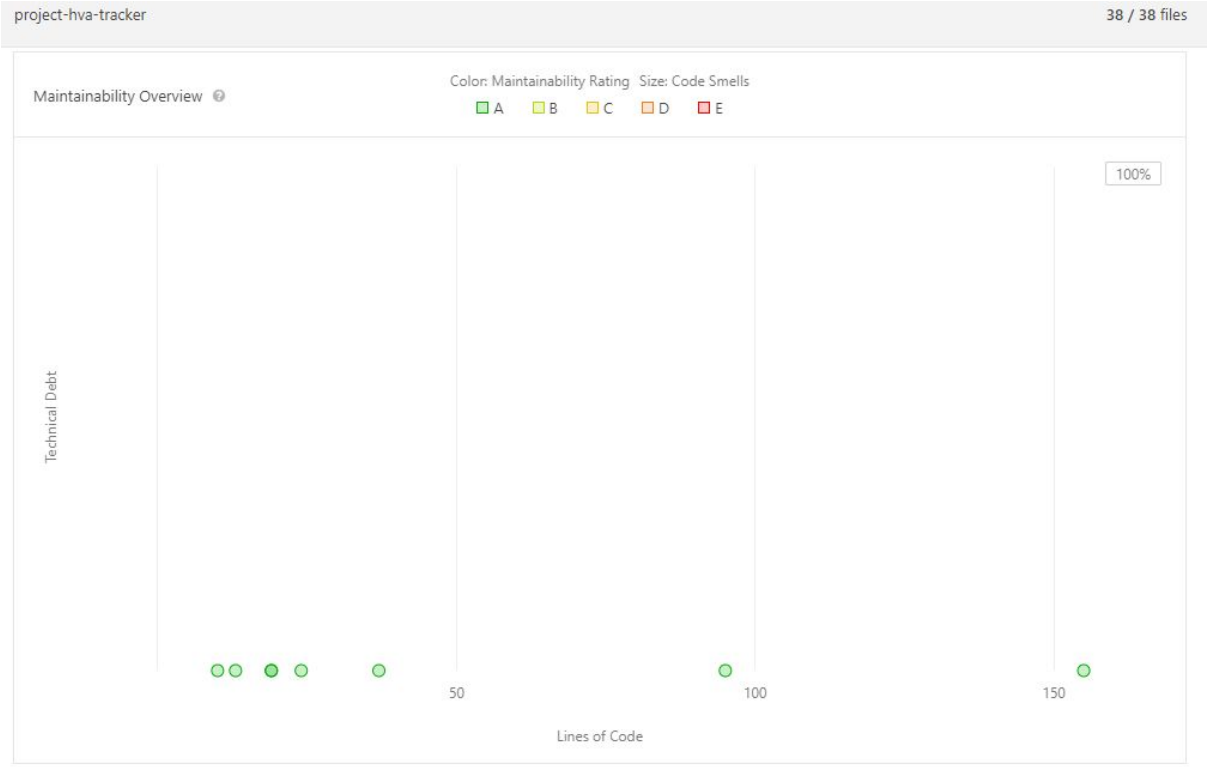


Maintainability

Tracker solution

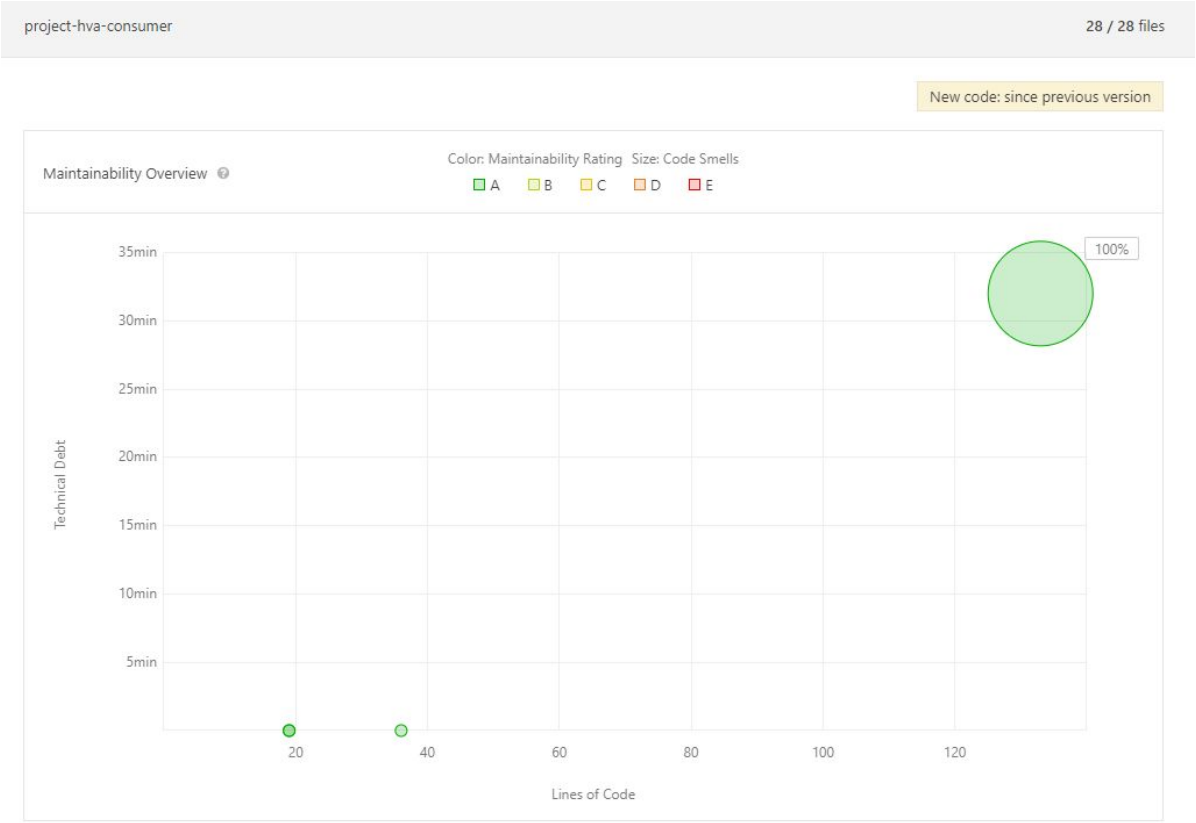
Tracker eerste run zag er zo uit:

De run na de unittests en het afronden van het project zag er zo uit:

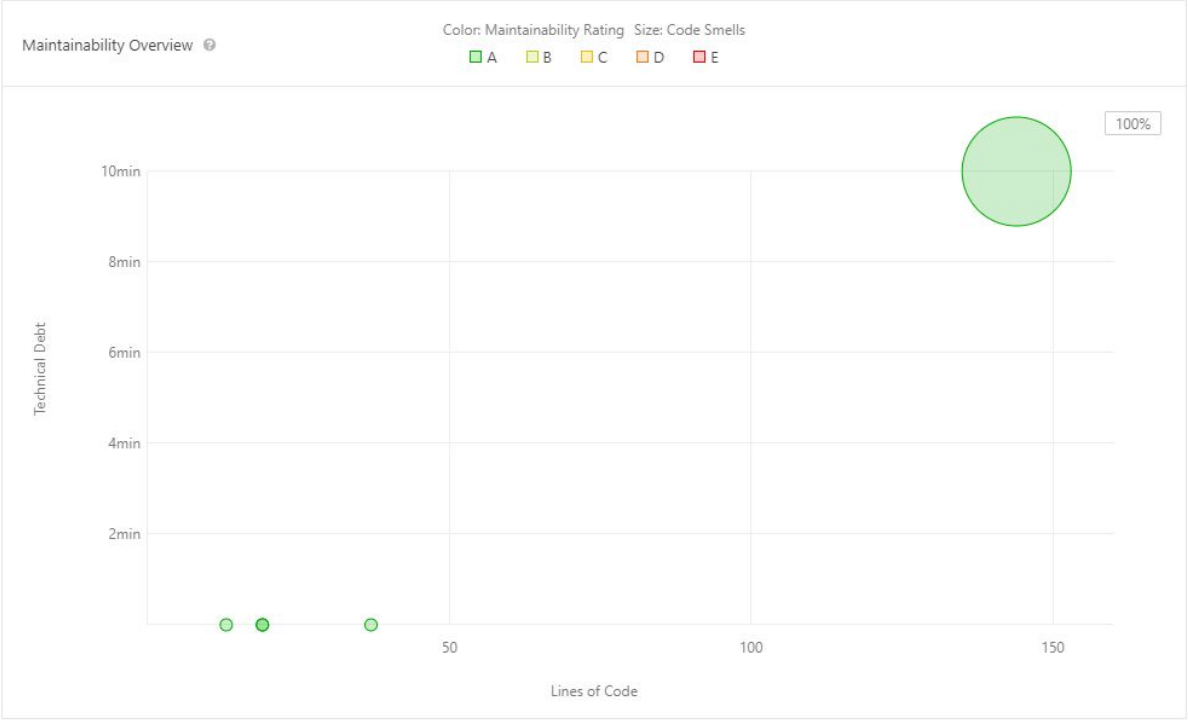


Consumer solution

Consumer eerste run zag er zo uit:



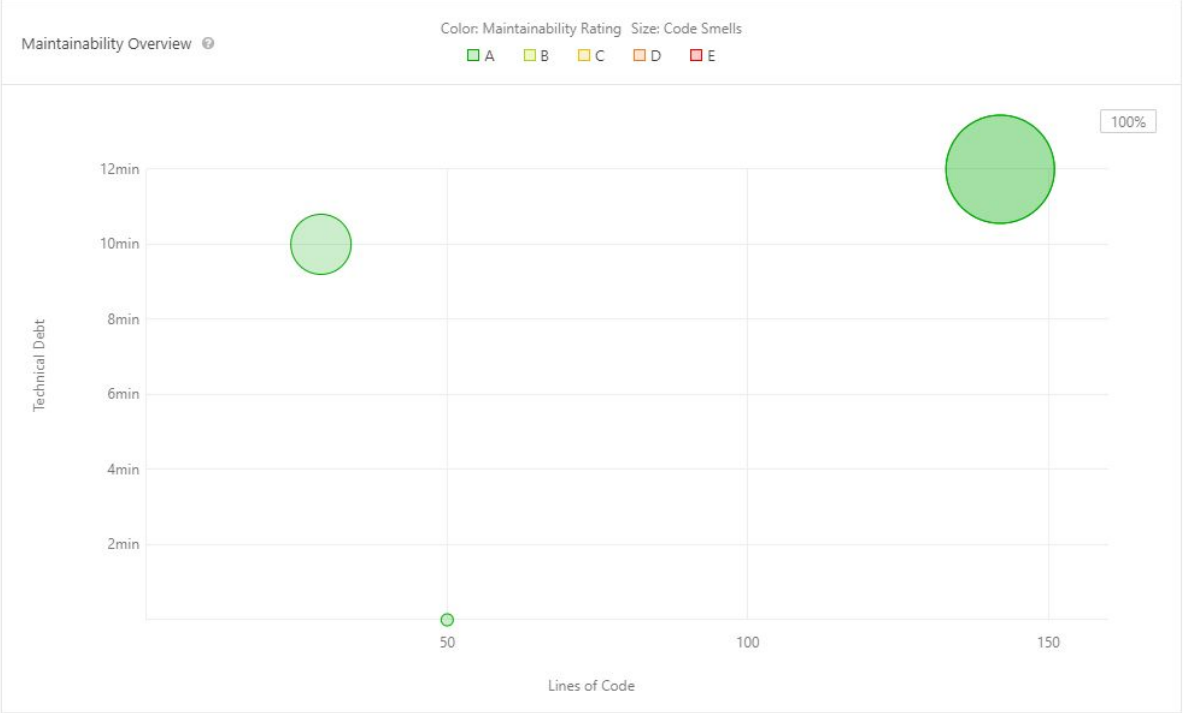
De run na de unittests en het afronden van het project zag er zo uit:



WebApi solution

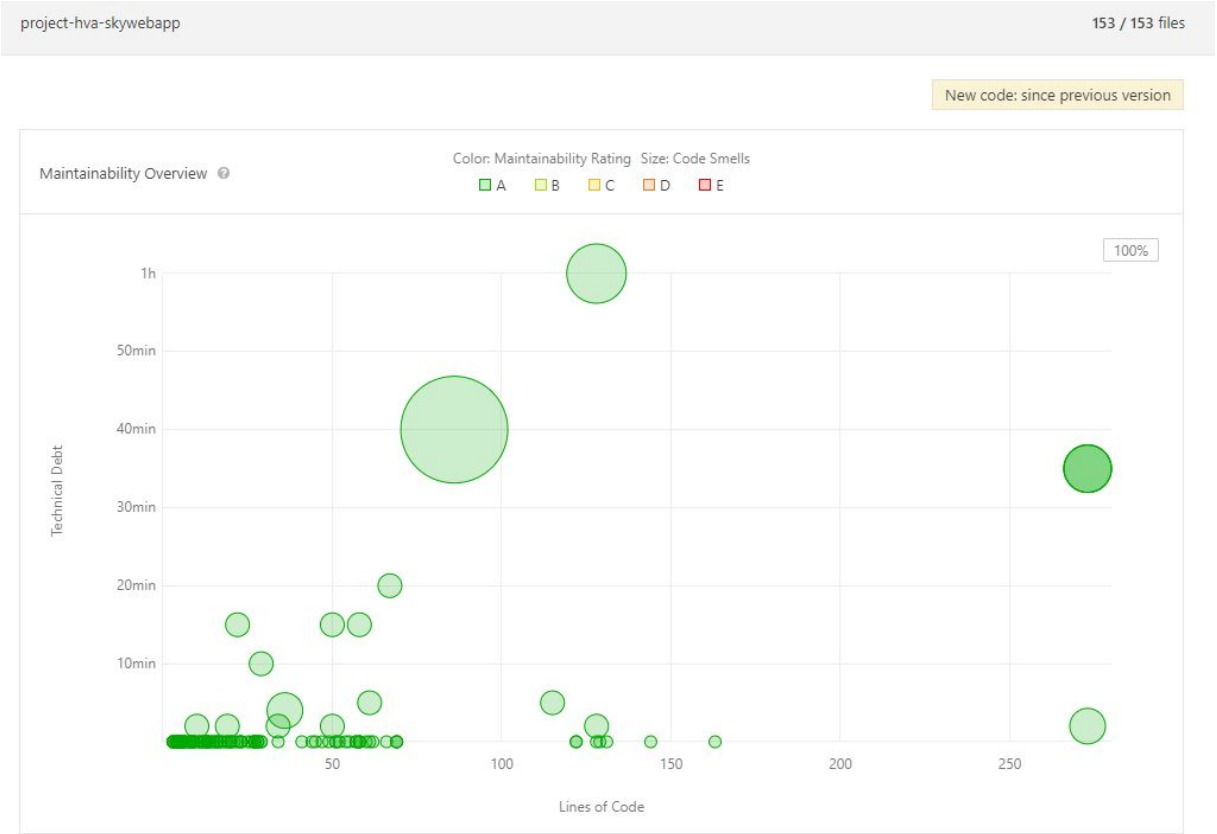
WebApi eerste run zag er zo uit:

De run na de unittests en het afronden van het project zag er zo uit:

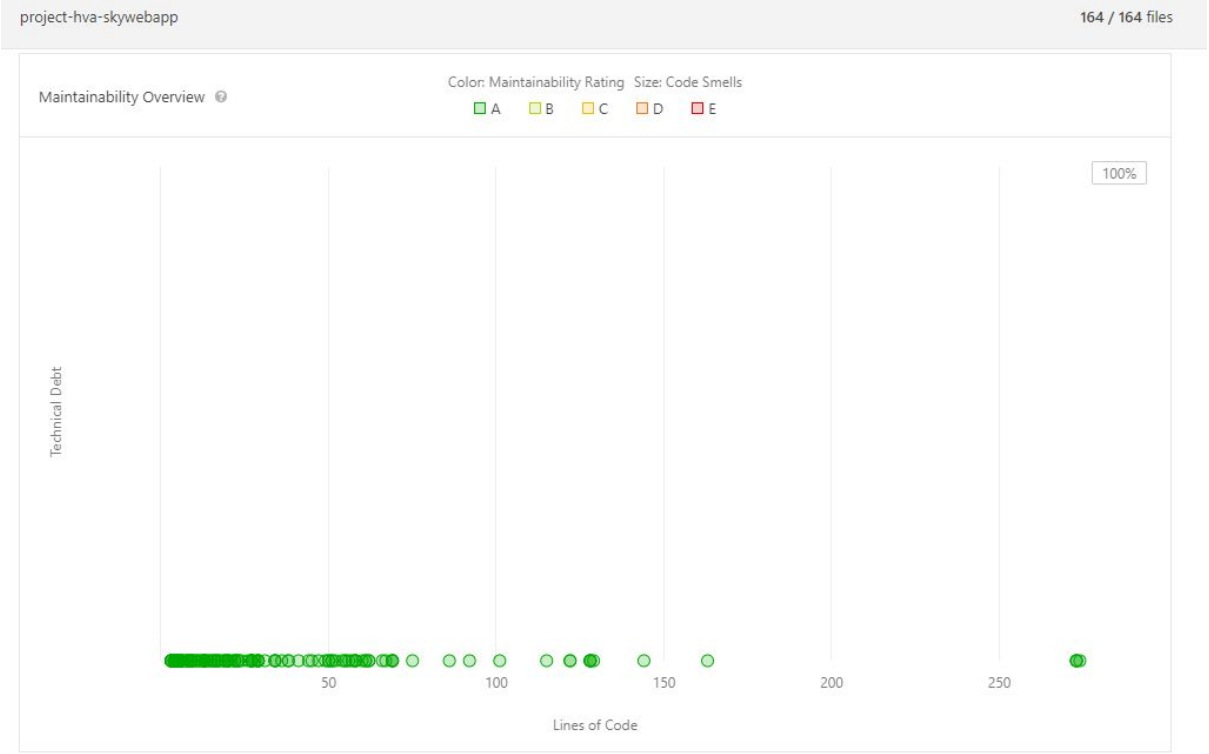


SkyWebApp solution

SkyWebApp eerste run zag er zo uit:



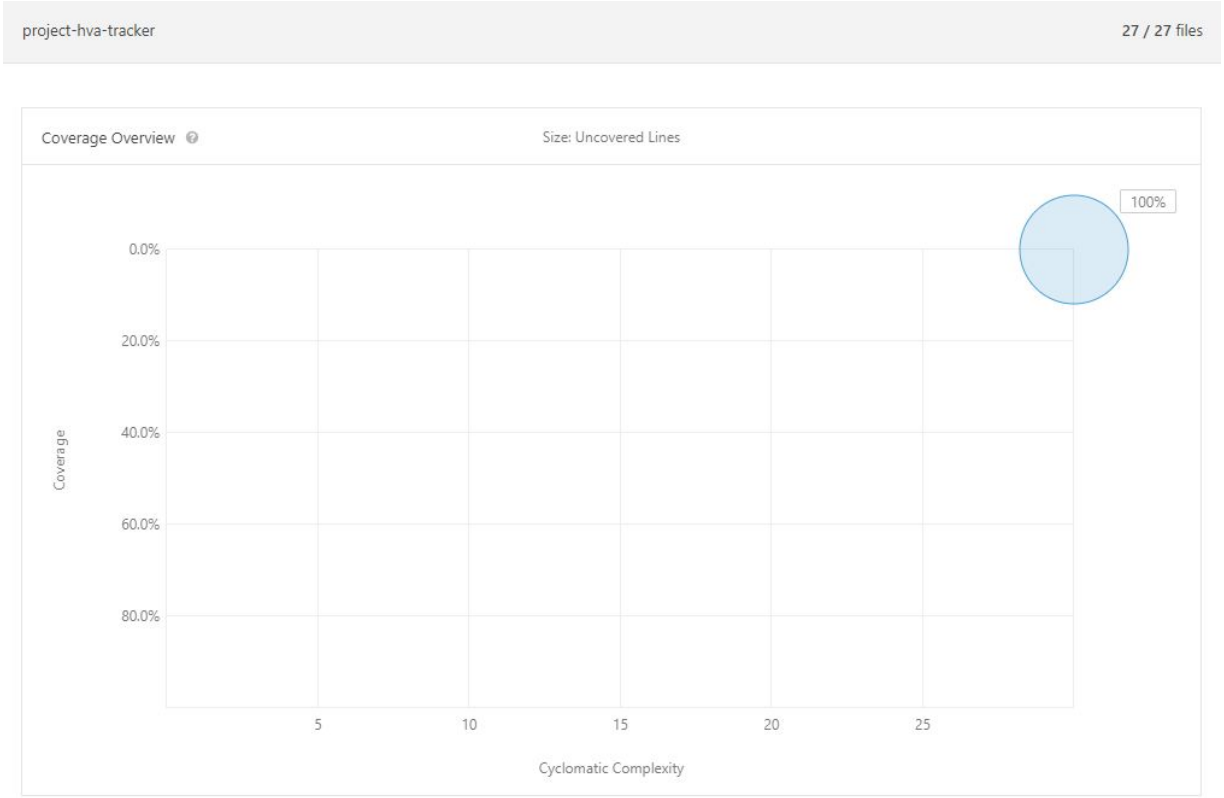
De run na de unittests en het afronden van het project zag er zo uit:



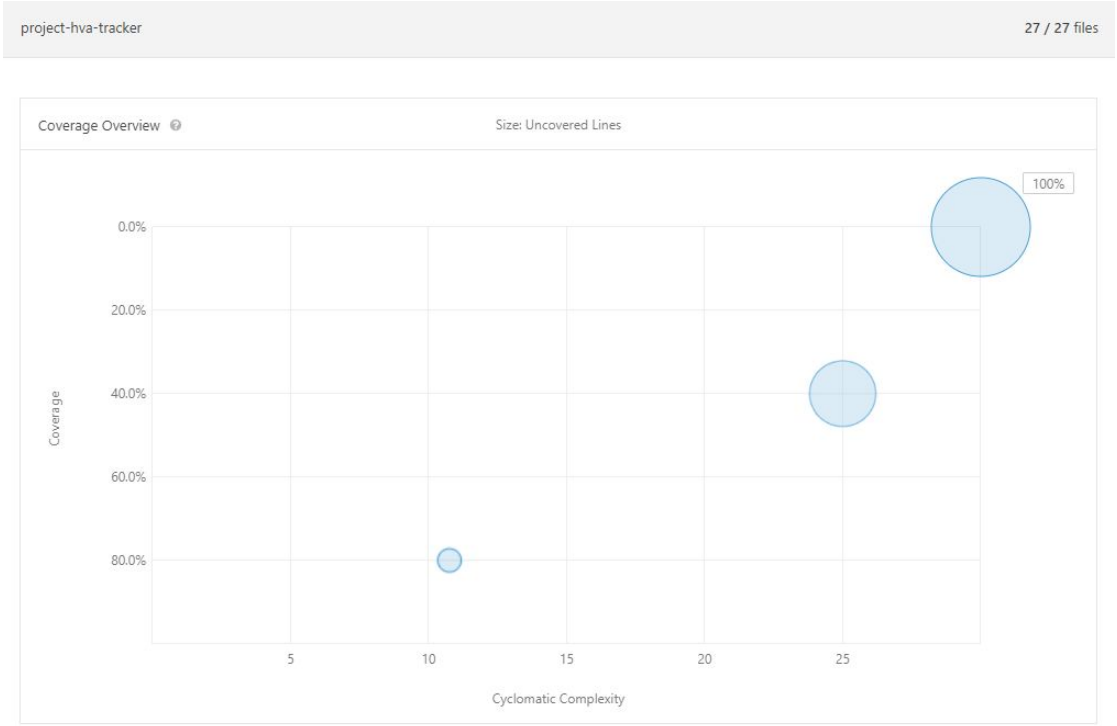
Coverage

Tracker solution

Before:

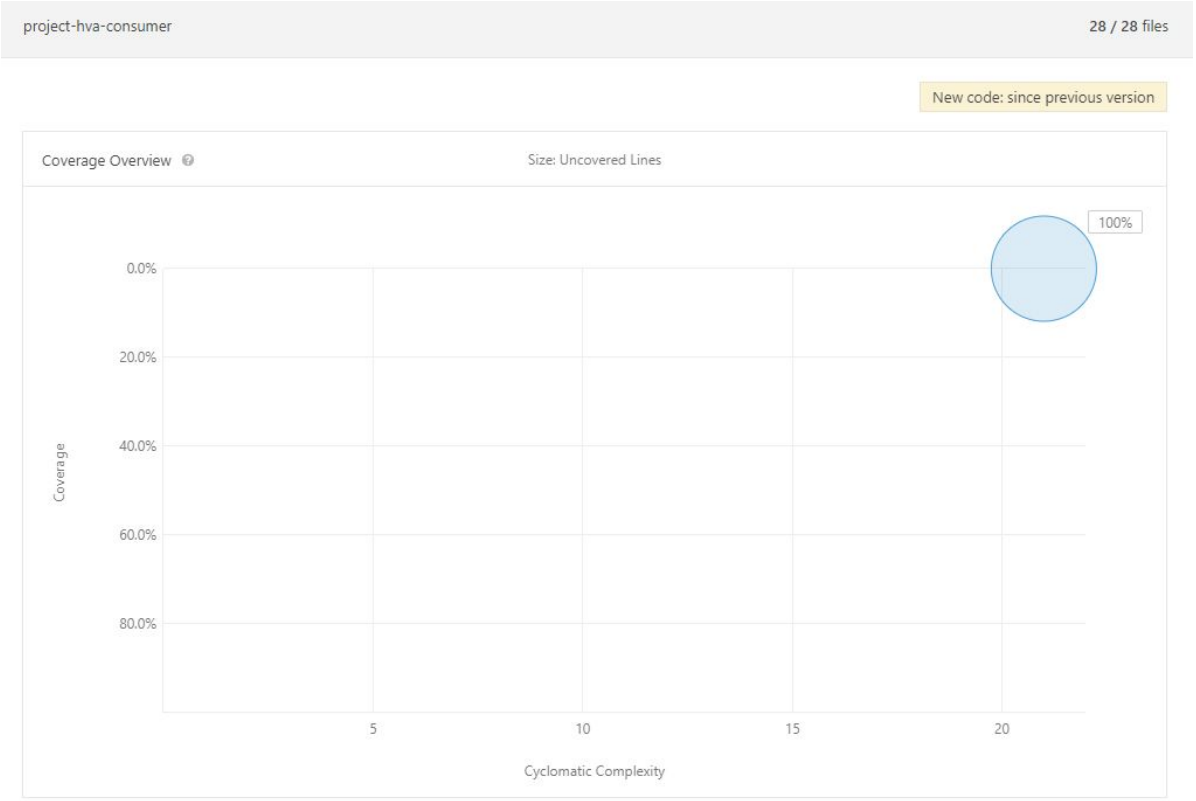


After:

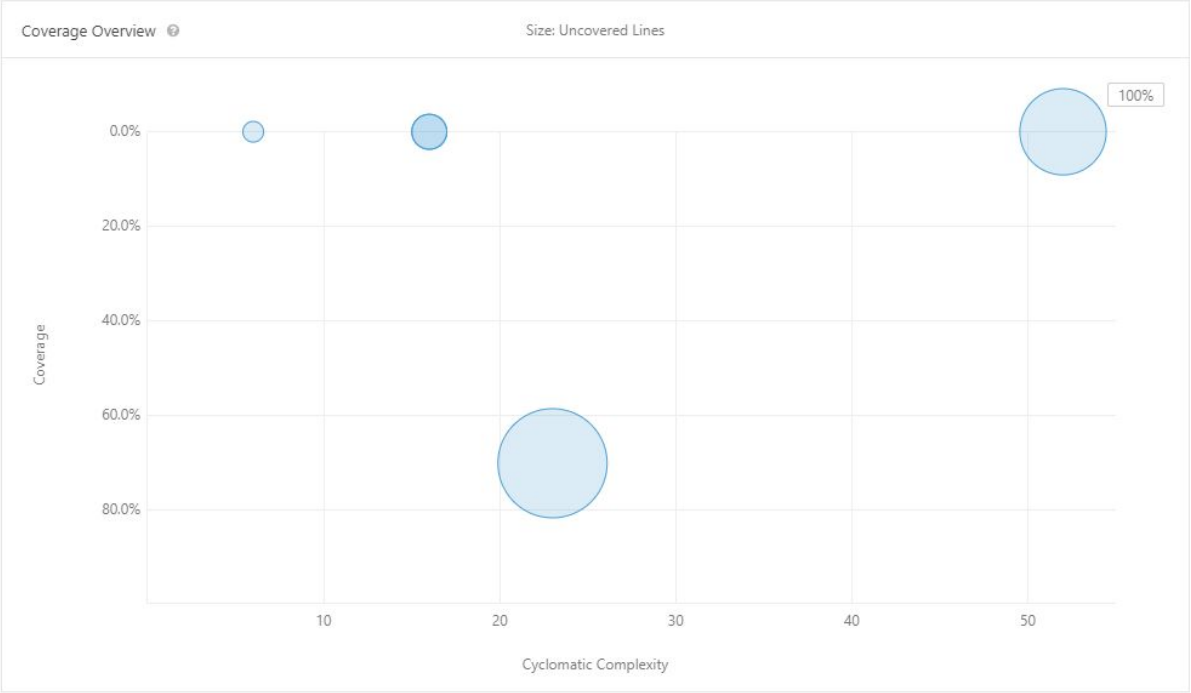


Consumer solution

before:

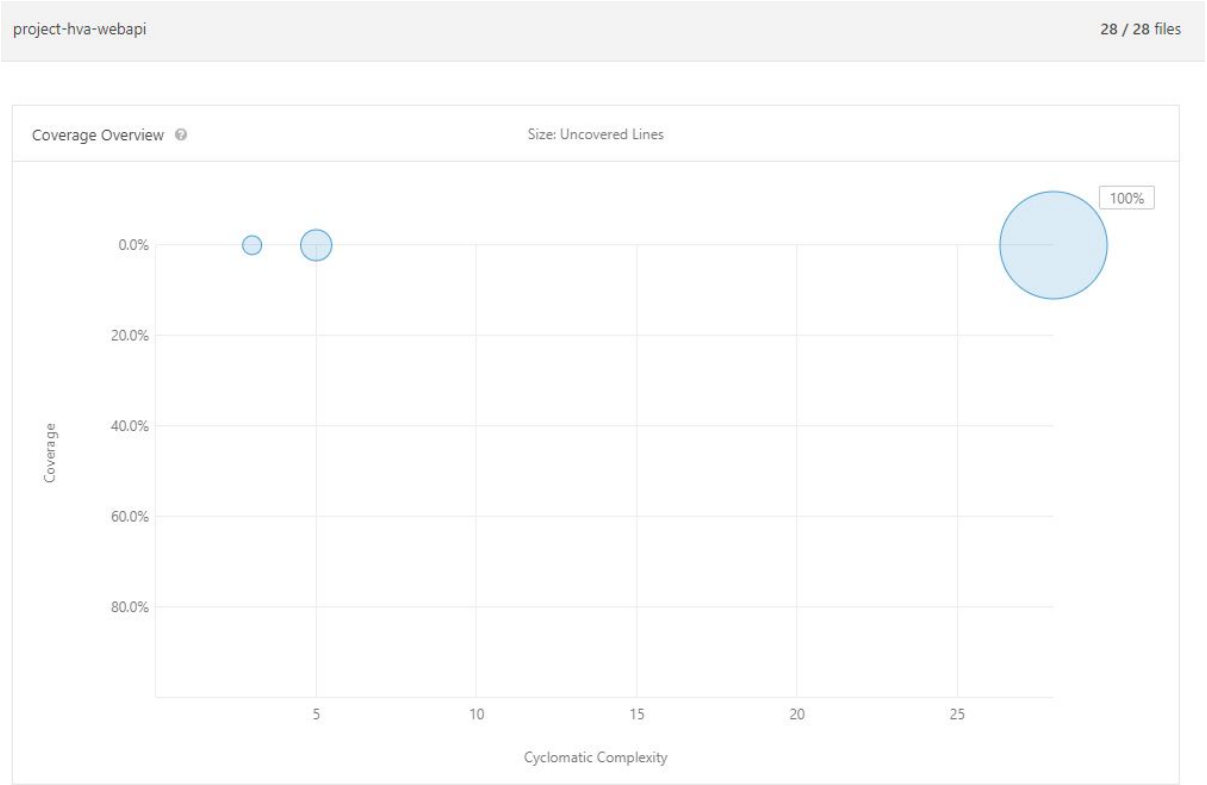


After:

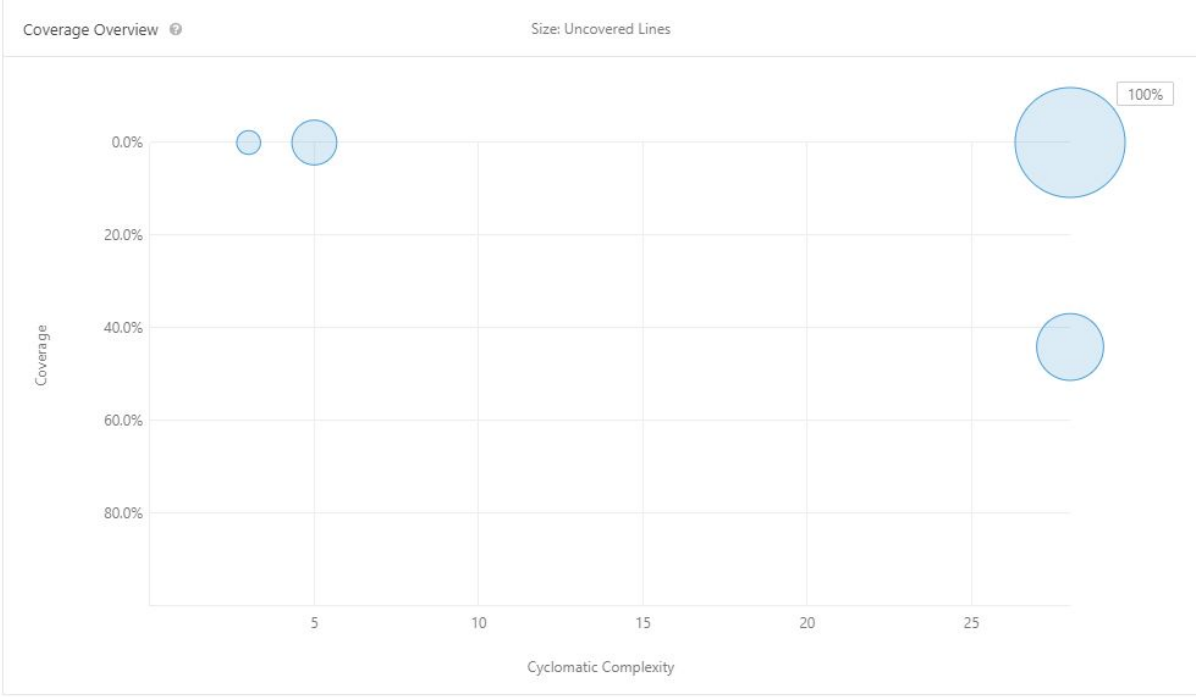


WebApi solution

Before:

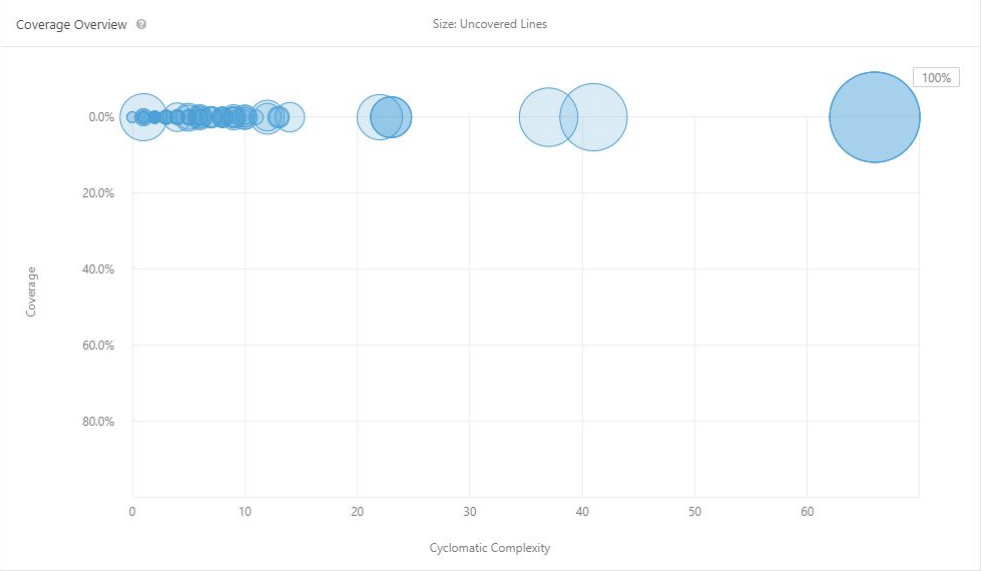


After:



SkyWebApp solution

Before:

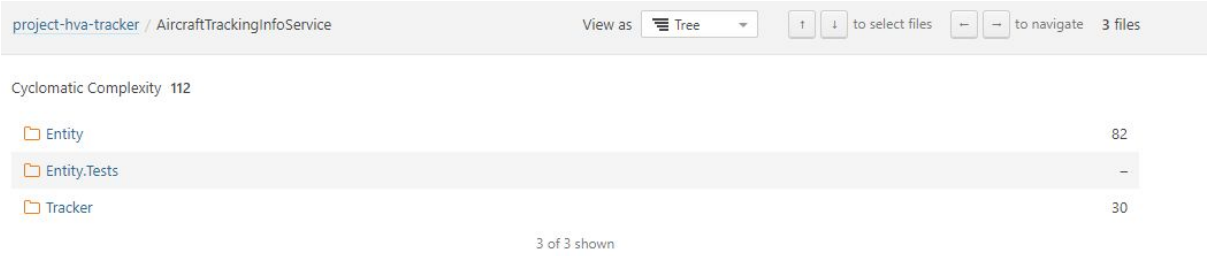


Na het afronden van het project is dit plaatje hetzelfde gebleven. Het is niet mogelijk om de Jest Unit Test die gemaakt is in SonarQube te krijgen, omdat Jest werkt vanuit de command line. SonarQube heeft een report nodig van de testen en bevindingen wat niet samengaat met een vanuit de command line te starten unit test.

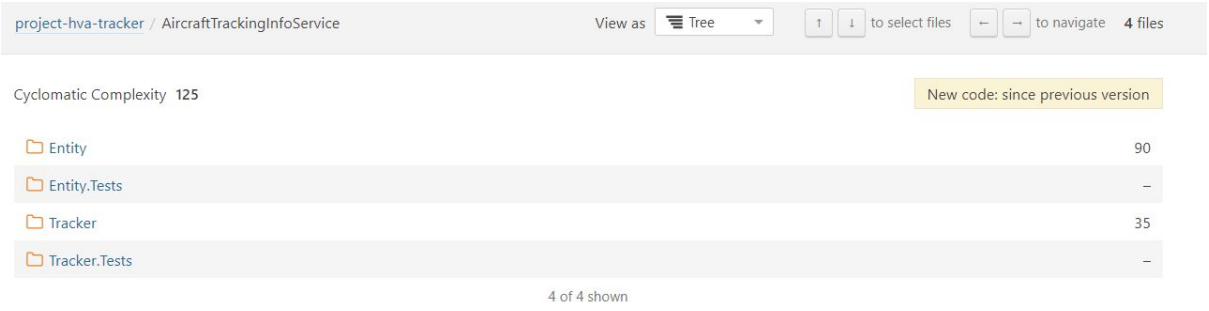
Cyclomatic complexity

Tracker solution

Before:

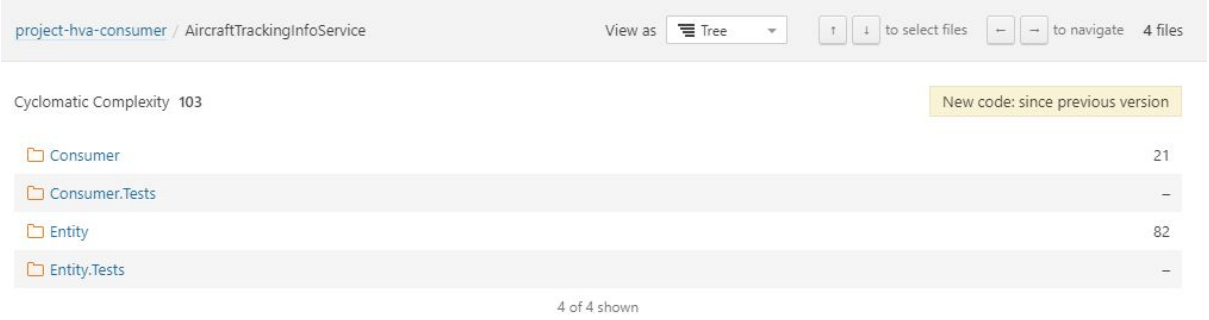


After:

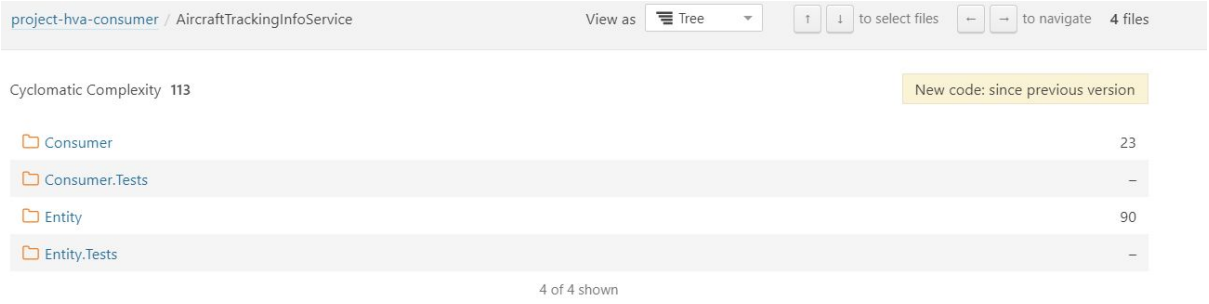


Consumer solution

Before:

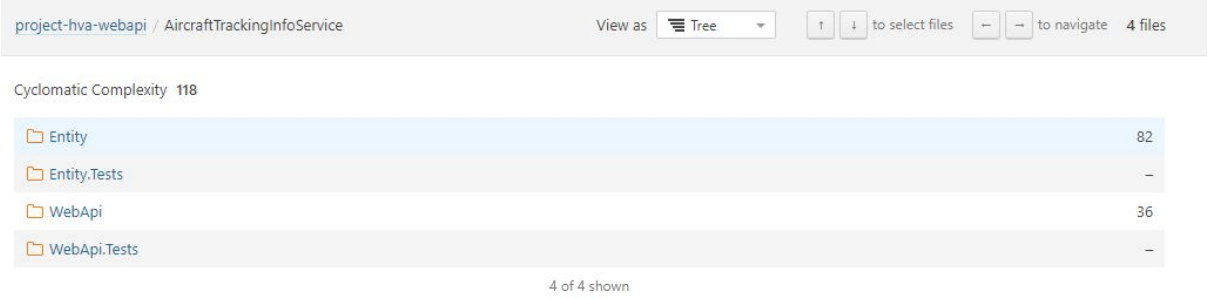


After:

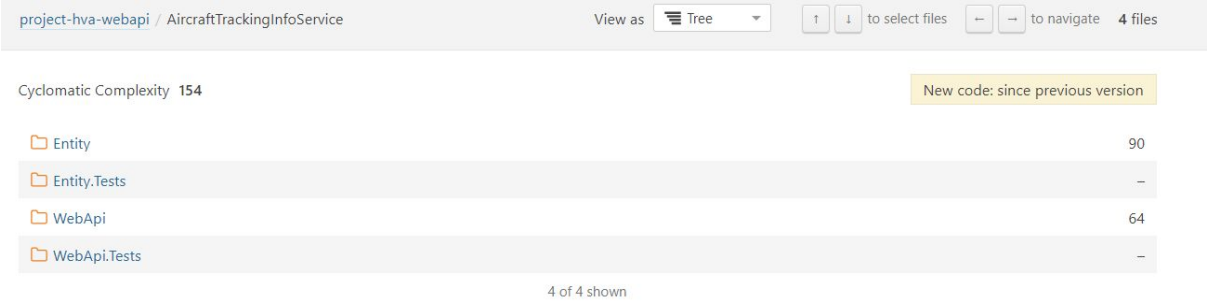


WebApi solution

Before:



After:



SkyWebApp solution

Before:

Cyclomatic Complexity 739 New code: since previous version

bindingFoundation	25
communicationFoundation	38
components	359
css	–
entity	11
generic	158
masterDetailFoundation	107
store	24
boot-client.tsx	5
boot-server.tsx	4
configureStore.ts	8
routes.tsx	0

12 of 12 shown

After:

project-hva-skywebapp View as Tree ↑ ↓ to select files ← → to navigate 10 files

Cyclomatic Complexity 892 New code: since previous version

__tests__/ClientApp/components/system/MockedAsset/component.Tests	2
ClientApp	849
Controllers	12
obj/Debug/netcoreapp2.2	–
Views	5
babel.config.js	0
Program.cs	3
Startup.cs	5
webpack.config.js	10
webpack.config.vendor.js	6

10 of 10 shown

Er is meer code geschreven voor de SkyWebApp en de complexiteit is toegenomen. Dit zit voornamelijk in de render functies van de verschillende pagina's. Deze bevatten allemaal een if-if-else constructie die bepaald of je in read-only, edit of new modus zit. Deze modus bepaald wat je te zien krijgt. Deze complexiteit lijkt dus heel diep, maar dit valt uiteindelijk mee.

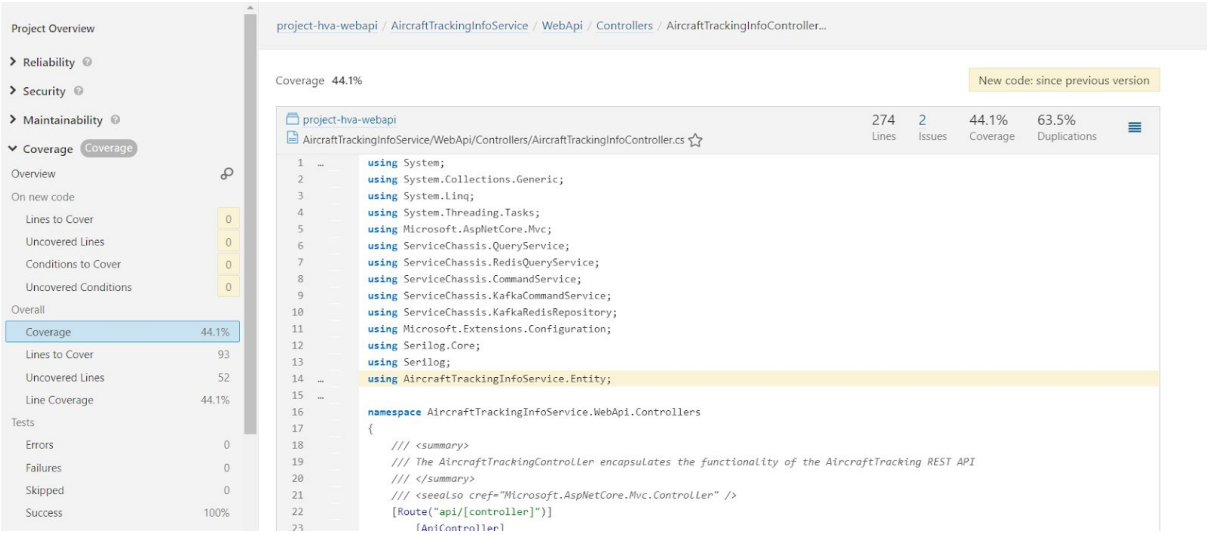
Assessment with Sonarqube including Unit Tests

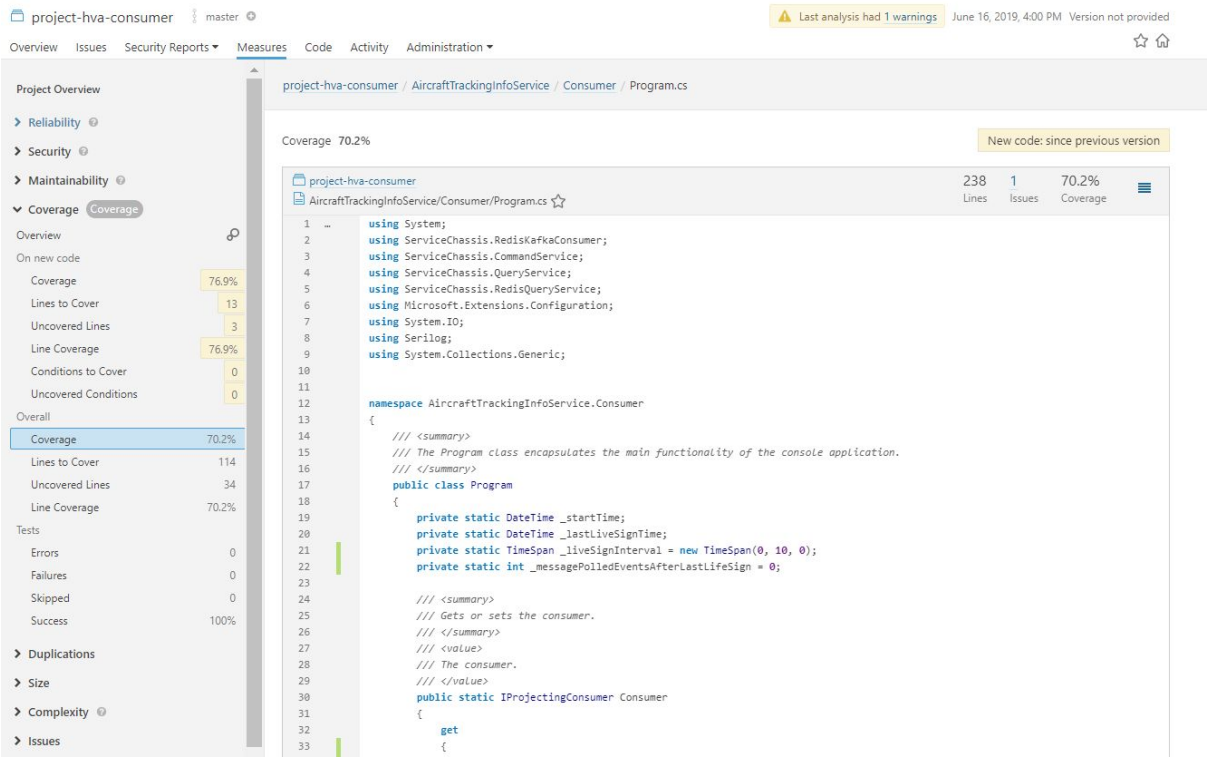
Om de Unit Tests in SonarQube weer te geven moest er heel wat configuratie plaatsvinden. Als Visual Studio de Unit Tests draait dan heeft deze een ingebouwde window voor de test results. Dit betekent dat er geen xml file wordt gegenereerd die SonarQube kan uitlezen. Datzelfde geldt ook voor het coverage percentage wat berekend wordt op het moment dat de Unit Tests worden uitgevoerd.

Na onderzoek hebben we op onderstaande manier Coverlet (NuGet Package) aan de gang gekregen om uiteindelijk toch het gewenste resultaat te krijgen in SonarQube. (voorbeeld is van WebApi)

```
dotnet add package coverlet.msbuild
dotnet test
WebApi.Tests/AircraftTrackingInfoService.WebApi.Tests.csproj
/p:CollectCoverage=true /p:CoverletOutputFormat=opencover
dotnet build-server shutdown
dotnet sonarscanner begin /k:"project-hva-webapi"
/d:sonar.host.url=http://localhost:9000
/d:sonar.cs.opencover.reportsPaths="WebApi.Tests\coverage.opencove
r.xml" /d:sonar.coverage.exclusions="**Tests*.cs"
dotnet build $pathName
dotnet sonarscanner end
```

Coverlet heeft een andere configuratie dan SonarQube en hier kwamen we niet volledig uit. We hebben hierdoor wel het ServiceChassis kunnen uitsluiten, maar niet de overige referenced projects (zoals Entity bijvoorbeeld). Daardoor is onze overall coverage op alle projecten vrij laag, want de tests geschreven voor deze referenced projects zijn niet meegenomen in de xml waardoor het lijkt alsof deze niet gecovered zijn. In de screenshots hieronder is te zien welke coverage we specifiek hebben op die solution.





Deze coverage is nog niet goed genoeg, want ze zijn nog niet boven de 80%, maar wel al een heel eind. In de meest ideale situatie is het percentage natuurlijk de volle 100%.

Unit Test

Aangezien bijna geen enkele van de requirements zoals ze geformuleerd zijn in opdracht 1 geschikt zijn voor het maken van unit tests, is besloten om hieronder een sommatie van de classes te geven waar we unit tests voor hebben geschreven.

1. Webapi.AircraftTrackingInfoController (Hier komen de webcalls binnen en worden ze gedistribueerd naar de juiste classes in het ServiceChassis)
2. Entity.AircraftArrivalInfo (Geeft de properties van een aankomst van een vliegtuig)
3. Entity.AircraftDepartureInfo (Geeft de properties van een vertrek van een vliegtuig)
4. Entity.AircraftTrackInfo (Geeft de properties van een aircraft)
5. Consumer.Program (Verzorgt de communicatie van Kafka naar Redis)
6. Tracker.Program (Haalt data van externe databron en vertaalt dit naar Entity object)
7. SkyWebAp (Verzorgt de front end)

In totaal komen uit deze classes de volgende Unit Tests. We hebben niet alle unit tests geprintgescreend, maar van elke solution minstens één.

Voor WebApi:

1. GetReturnsNull()
2. gettopiccountedlistreturnssortedListOrNull()
3. PutReturnsReplaced()
4. PutReturnsError()

Voor Consumer:

5. RunMain_WithoutArgs_ShouldNotThrow()
6. RunConsumerMessageRecieved_WithValidEventArgs_ShouldNotThrow()
7. RunConsumerMessageRecieved_WithNullEventArgs_ShouldNotThrow()
8. RunConsumerMessageRecieved_WithDataNullEventArgs_ShouldNotThrow()

Voor Entity:

9. SetAircraftTrackObject_IsEqualToGetAircraftTrackObject_ShouldNotThrow()
10. SetAircraftTrackObject_IsEqualToDifferentGetArrivalTrackObject_ShouldThrow()
11. SetNullAircraftTrackObject_IsEqualToAnotherGetArrivalTrackObject_ShouldThrow()
12. SetDepartureTrackObject_IsEqualToGetArrivalTrackObject_ShouldNotThrow()
13. SetDepartureTracObject_IsEqualToDifferentGetArrivalTrackObject_ShouldThrow()
14. SetNullDepartureTrackObject_IsEqualToOtherGetArrivalTrackObject_ShouldThrow()
15. SetArrivalTrackObject_IsEqualToGetArrivalTrackObject_ShouldNotThrow()
16. SetArrivalTracObject_IsEqualToDifferentGetArrivalTrackObject_ShouldThrow()
17. SetNullArrivalTrackObject_IsEqualToOtherGetArrivalTrackObject_ShouldThrow()

Voor SkyWebApp:

18. CheckIsHeading_IsMockedAssets_shouldBeEqual()

Voor Tracker:

19. CallRestMethod_ReturnsString()
20. GetConfig_ShouldReturnConfig()
21. GetTrackerData_WithNoRecordsFound_ShouldReturnEmptyEntitiesList()
22. getArrivalDataShouldReturnJsonWithArrivalTime()
23. getDepartureDataShouldReturnJsonWithArrivalTime()

First en Right BICEP

De verschillende unit tests voldoen aan alle onderstaande eisen. Zij zijn op dezelfde manier opgebouwd waardoor zij voldoen aan de FIRST en Right BICEP principes.

De tests zijn **Fast**, want alle tests zijn binnen één seconde afgerond. Het gemiddelde ligt rond de 600 ms per testset wat een prima resultaat is.

De tests zijn **Isolated** want iedere test doet de 3 A's, Arrange, Act en Assert, waarbij er geen gebruik wordt gemaakt van services of omgevingen buiten de testklasse en er zijn geen volgordelijkheidsafhankelijkheden.

De tests zijn **Repeatable** want ze zijn niet afhankelijk van andere code, er zijn geen datum- of timestampsafhankelijkheden die de uitkomst kunnen beïnvloeden en in het arrange-gedeelte van de tests wordt alle data geïnitialiseerd.

De tests zijn **Self-validating**. De methode <<waarde>>should.be(<<waarde>>) komt van de Fluent Assertions Library en geeft een exception als de vergeleken waarden niet met elkaar overeenkomen. Daarnaast kunnen we testen op verschillende error messages. Dus als er een exception moet ontstaan, omdat een waarde bijvoorbeeld niet null mag zijn, kunnen we ook testen op <<waarde>>.should().Throw(); zonder dat de tests niet slaagt.

Onze tests zijn **Thorough** want we testen alle mogelijke alle grensgevallen. Het grensgeval van een ander soort type data, bijvoorbeeld of een integer wel een integer is, wordt niet getest. Doordat er frontend met Typescript wordt gewerkt is dit overbodig. Typescript is typed language wat ervoor zorgt dat er frontend al afgevangen wordt of een waarde wel van een bepaald type is.

Onze unit tests voldoen daarnaast aan het Right BICEPS principe, omdat we alleen resultaten testen (**Results**), namelijk de waarden van de attributen, en niet de werking van een methode.

We hebben alle grensgevallen (**Boundaries**) gecheckt. Zo checken we op null waarden.

Inverse relationships zijn hier niet van toepassing en daar hoeft dus ook niet op getest te worden. **Cross checking** van de testresultaten is, gezien de aard van de klasse niet van toepassing. We testen of een waarde juist is ge-set. Alleen door een vergelijking van het resultaat met de verwachting kan je dit testen.

Error conditions worden geforceerd, want een exception wordt ge-throwed als de test negatief is.

Gezien de snelheid van de tests, gemiddeld 600 ms per test, zijn de **Performance characteristics** naar behoren.

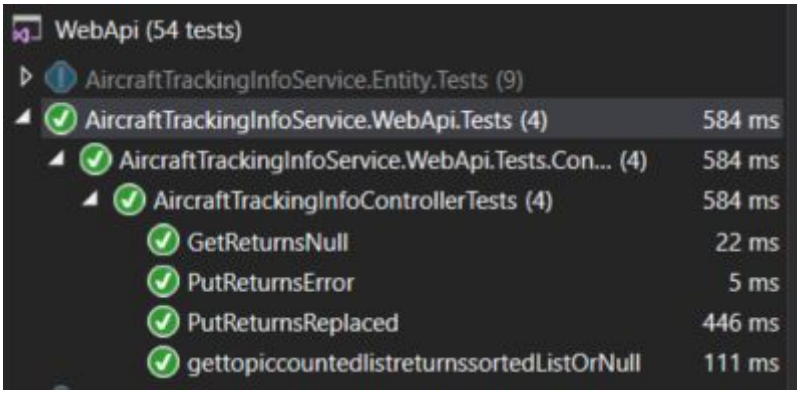
WebApi AircraftTrackingInfoControllerTests

De AircraftTrackingInfoController is de klasse waar de resultaten van de HTTP request worden ontvangen en middels de repository doorgezet naar de juiste klassen in de ServerChassis. AircraftTrackingInfoController bevat de methodes get, getAudit, getTopicCountedList, getTopicNavigatableEntity, post, put en delete. Alleen de methodes getTopicCountedList en put doen nog iets meer dan alleen het http request ontvangen en doorgeven en heeft een unit test ook daadwerkelijk toegevoegde waarde. We hebben daarom alleen voor deze en de get-methode unittests geschreven. De unit test voor de get methode is geschreven voor academische redenen.

De GetCountedTopicList methode checked of er de teruggekomen lijst null is of een of een specifieke waarde ervan null is. Daarnaast sorteert deze methode de teruggekomen lijst.

De methode Put geeft aan of de veranderde waarde goed is doorgegeven. Zo niet, dan geeft de methode “error” aan.

Wij hebben de klasse AircraftTrackingInfoController volgen de principes van F.I.R.S.T. en Right BICEPS getest.




```
[Fact]
[Trait("Category", "Command")]
[Trait("Category", "Post")]
- references
public void gettopiccountedlistreturnssortedListOrNull()
{
    // Arrange
    Mock<AircraftTrackInfo> airCraftTrackInfoMock1 = new Mock<AircraftTrackInfo>();
    Mock<AircraftTrackInfo> airCraftTrackInfoMock2 = new Mock<AircraftTrackInfo>();
    Mock<AircraftTrackInfo> airCraftTrackInfoMock3 = new Mock<AircraftTrackInfo>();

    airCraftTrackInfoMock1.Setup(foo => foo.RegistrationNumber).Returns("1");
    airCraftTrackInfoMock2.Setup(foo => foo.RegistrationNumber).Returns("2");
    airCraftTrackInfoMock3.Setup(foo => foo.RegistrationNumber).Returns("3");


    List<dynamic> list = new List<dynamic>();
    list.Add(airCraftTrackInfoMock2.Object);
    list.Add(airCraftTrackInfoMock1.Object);
    list.Add(airCraftTrackInfoMock3.Object);
    List<dynamic> sortedList = new List<dynamic>();
    sortedList.Add(airCraftTrackInfoMock1.Object);
    sortedList.Add(airCraftTrackInfoMock2.Object);
    sortedList.Add(airCraftTrackInfoMock3.Object);
    //Act test sorting
    repositoryMock.Setup(foo => foo.GetTopicCountedList(null, null)).Returns(new TopicCountedList<dynamic>(list));
    var result = _aircraftTrackingInfoController.GetTopicCountedList(null, null);
    Assert.Equal(result.List, sortedList);

    //Act test no TopicCountedList
    repositoryMock.Setup(foo => foo.GetTopicCountedList(null, null)).Returns((TopicCountedList<dynamic>)(null));
    var noTopicCountedList = _aircraftTrackingInfoController.GetTopicCountedList(null, null);
    Assert.Null(noTopicCountedList);
```

Entity AircraftTrackInfoTests, AircraftArrivalInfoTests en AircraftDepartureInfoTests

AircractTrackingInfo, AircraftArrivalInfo en AircraftDepartureInfo zijn klassen die informatie over (onderdelen van) een aircraft bevatten. De tests voor alle drie de klassen zijn op dezelfde manier opgebouwd. De eerste test test of een object van de klasse de waarden inderdaad heeft die middels de set-methode aan het object zijn meegegeven. De tweede functie test of een tweede object van de klasse dezelfde attribuutwaarden heeft aan het eerste object. En de derde test test of er een exceptie wordt gegenereerd als de attribuutwaarden van het object null zijn.

Onze drie tests zijn goede tests, want ze voldoen aan de F.I.R.S.T. en Right BICEPS principes zoals eerder zijn beschreven. Er is hier geen sprake van Inverse Relationships, want er worden geen berekeningen gemaakt. We kunnen enkel testen of de results overeenkomen op het moment dat de waarde met set en get hetzelfde resultaat oplevert.

AircraftTrackingInfoService.Entity.Tests (9)	702 ms
Entity.Tests (9)	702 ms
AircraftArrivalInfoTests (3)	236 ms
SetArrivalTracObject_IsEqualToDifferentGetArrivalTrackObject_ShouldThrow	4 ms
SetArrivalTrackObject_IsEqualToGetArrivalTrackObject_ShouldNotThrow	2 ms
SetNullArrivalTrackObject_IsEqualToOtherGetArrivalTrackObject_ShouldThrow	230 ms
AircraftDepartureInfoTests (3)	228 ms
SetArrivalTracObject_IsEqualToDifferentGetArrivalTrackObject_ShouldThrow	200 ms
SetArrivalTrackObject_IsEqualToGetArrivalTrackObject_ShouldNotThrow	24 ms
SetNullArrivalTrackObject_IsEqualToOtherGetArrivalTrackObject_ShouldThrow	4 ms
AircraftTrackTests (3)	238 ms
SetAircraftTrackObject_IsEqualToDifferentGetArrivalTrackObject_ShouldThrow	4 ms
SetAircraftTrackObject_IsEqualToGetAircraftTrackObject_ShouldNotThrow	4 ms
SetNullAircraftTrackObject_IsEqualToAnotherGetArrivalTrackObject_ShouldThrow	230 ms

```
/// <summary>
/// Constructs AircraftTrackInfo object and test properties.
/// </summary>
[Fact]
[Trait("Category", "Entity")]
[Trait("Category", "ConstructorAndProperties")]

0 references
public void SetAircraftTrackObject_IsEqualToDifferentGetArrivalTrackObject_ShouldThrow()
{

    // Arrange
    var differentFakeAircraftTrack = new AircraftTrackInfo();

    var fakeAircraftTrack = new AircraftTrackInfo
    {
        Id = "AircraftTrack_1",
        Uuid = "Test",
        IataCodeDeparture = "Test",
        IataCodeArrival = "Test",
        FightNumber = "Test",
        IataCodeAirline = "Test",
        RegistrationNumber = "Test",
        Latitude = "Test",
        Longitude = "Test",
        Altitude = "Test",
        Direction = "Test",
        Status = "Test"
    };

    // Act
    Action act = () =>
    {
        differentFakeAircraftTrack.Id.Should().Be("AircraftTrack_1", "because it's what was set before.");
        differentFakeAircraftTrack.Uuid.Should().Be("Test", "because it's what was set before.");
        differentFakeAircraftTrack.IataCodeDeparture.Should().Be("Test", "because it's what was set before.");
        differentFakeAircraftTrack.IataCodeArrival.Should().Be("Test", "because it's what was set before.");
        differentFakeAircraftTrack.IataCodeAirline.Should().Be("Test", "because it's what was set before.");
        differentFakeAircraftTrack.FightNumber.Should().Be("Test", "because it's what was set before.");
        differentFakeAircraftTrack.RegistrationNumber.Should().Be("Test", "because it's what was set before.");
        differentFakeAircraftTrack.Latitude.Should().Be("Test", "because it's what was set before.");
        differentFakeAircraftTrack.Longitude.Should().Be("Test", "because it's what was set before.");
        differentFakeAircraftTrack.Altitude.Should().Be("Test", "because it's what was set before.");
        differentFakeAircraftTrack.Direction.Should().Be("Test", "because it's what was set before.");
        differentFakeAircraftTrack.Status.Should().Be("Test", "because it's what was set before.");
    };
    // Assert
    act.Should().Throw<XunitException>();
}
```

Consumer ProgramTests

De consumer haalt data uit Apache Kafka en transformeert dit naar een Redis key/object. De methoden geven terug of dit gelukt is door middel van Logging. Deze logger wordt gemockt en tests of dit geen exception teruggeeft. Daarnaast is per methode een vast logging bericht dus ook deze worden getest door middel van een .should().be(...
Deze tests zijn dus niet afhankelijk van andere code of objecten die terugkomen. Als deze methoden worden aangeroepen dienen zij alleen een logging message terug te geven die gelijk is aan het van te voren verwachtte testresultaat.

Omdat er meerdere tests zijn die gebruik maken van deze logger heb ik lokaal een functie geschreven die een gemockte Logger teruggeeft. Deze uitkomsten worden niet daadwerkelijk verstuurd naar Logstash.

AircraftTrackingInfo.Consumer (61 tests)	
AircraftTrackingInfoService.Consumer.Tests (4)	616 ms
AircraftTrackingInfoService.Consumer.Tests (4)	616 ms
ProgramTests (4)	616 ms
RunConsumerMessageRecieved_WithDataNullEventArgs_ShouldNotThrow	347 ms
RunConsumerMessageRecieved_WithNullEventArgs_ShouldNotThrow	2 ms
RunConsumerMessageRecieved_WithValidEventArgs_ShouldNotThrow	2 ms
RunMain_WithoutArgs_ShouldNotThrow	265 ms

```
/// <summary>
/// Runs the consumer message recieved event handler with a valid event arguments.
/// </summary>
[Fact]
[Trait("Category", "Consumer")]
[Trait("Category", "ConsumerMessageRecieved")]
0 references
public void RunConsumerMessageRecieved_WithValidEventArgs_ShouldNotThrow()
{
    // Arrange
    object stubSender = null;
    var e = new MessageReceivedEventArgs();
    e.Data = "Test";
    var stubLogger = CreateDefaultLogger();

    // Act
    Action act = () =>
    {
        Program.Logger = stubLogger;
        Program.Logger = new LoggerConfiguration().WriteTo.Console().CreateLogger();
        Program.ConsumerMessageRecieved(stubSender, e);
    };

    // Assert
    act.Should().NotThrow();
}
```

Tracker ProgramTests

De Tracker solution haalt data op van twee externe API's. Er wordt een REST-call gedaan waar een AircraftTrackInfo object uitgehaald kan worden en daarna wordt met deze data een tweede en een derde call gedaan naar een andere API om de DepartureInfo en ArrivalInfo op te halen en deze data aan het juiste object te hangen. We testen of er een string terugkomt op het moment dat we de REST-call doen. Dit kunnen we uitvoeren doordat de REST-call in een aparte class zit en we de daadwerkelijke call kunnen mocken. Ook testen we of het toevoegen van data aan een al bestaand object goed gaat door een Should().BeEqual() te gebruiken voor een testobject die we in de functie stoppen die data toevoegt aan een bestaand object. Ook testen we wat er gebeurd als er geen data terugkomt van de REST-call, daarvoor hieronder het voorbeeld.

```
[Fact]
0 references
public void GetTrackerDataWithNoRecordsFoundShouldReturnEmptyEntitiesList()
{

    //Arrange

    Mock<IGetTrackerData> trackerDataGetter = new Mock<IGetTrackerData>();
    trackerDataGetter.Setup(foo => foo.CallRestMethod(It.IsAny<string>())).Returns("No Record Found");
    AircraftTrackingInfoSeedData aircraftTrackingInfoSeedData = new AircraftTrackingInfoSeedData(trackerDataGetter.Object);

    //Act
    var result = aircraftTrackingInfoSeedData.GetTrackerData();


    //Assert
    Assert.Empty(result);

}
```

AircraftTrackingService.Tracker.Tests (5)	836 ms
AircraftTrackingService.Tracker.Tests (4)	582 ms
AircraftTrackingInfoSeedDataTests (3)	470 ms
GetTrackerDataWithNoRecordsFoundShouldReturnEmptyEntitiesList	246 ms
getArrivalDataShouldReturnJsonWithArrivalTime	118 ms
getDepartureDataShouldReturnJsonWithArrivalTime	106 ms
TrackerDataGetterTests (1)	112 ms
CallRestMethodReturnsString	112 ms
Tracker.Tests (1)	254 ms
ProgramTests (1)	254 ms
GetConfigShouldReturnConfig	254 ms

SkyWebApTests

Deze test test of de gehele entities op het scherm wordt getoond.

```
// test Render from MockedAssetCollectionContainer
import React from 'react';
import { render, cleanup } from '@testing-library/react';
import configureMockStore from "redux-mock-store"
import * as cachingMM from 'masterDetailFoundation/CachingMasterManager';
import 'jest-dom/extend-expect'
import MockedAssetCollectionContainer from 'components/system/MockedAsset/component/MockedAssetCollectionContainer';


const mockCachingMasterManager = jest.fn();

cachingMM.CachingMasterManager = jest.fn().mockImplementation(() => {
    return {
        loadList: mockCachingMasterManager
    };
});

// automatically unmount and cleanup DOM after the test is finished.
afterEach(cleanup);

test('Check is Heading is MockedAssets', () => {

    const mockStore = configureMockStore();

    const store = mockStore({});
    const match = { 'params': { 'pageNumber': 2 } };

    const entities = [{ Id: 1, Uuid: 2, IataNumber: 3 }];

    const { container, getByText } = render(<MockedAssetCollectionContainer match={match} store={store}
        entities={entities} />);


    expect(container.getElementsByClassName("table")[0].getElementsByTagName
        ("tbody")[0].childNodes.length).toEqual(entities.length)

});
```

```
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        21.187s
Ran all test suites.

C:\Users\Siree\Mijn documenten\VisualStudioProjects\AircraftTracker\SkyWebApp\Sk
yWebApp>
```

Pipeline in Azure DevOps

Tijdens het project hebben we gewerkt met TFS, Team Foundation Server, van Azure DevOps. Dit werkt ongeveer zoals Git, behalve dat het veel uitgebreider is. Scrum board, Repos, en pipelines kunnen ook binnen TFS opgezet worden. De Pipeline die wij gebouwd hebben zit in TFS en doet continuous integration. Hier zit een agent in die jobs kan verwerken.

Telkens als iemand incheckt dan wordt er een nieuwe job geproduceerd en binnen die job wordt opnieuw gebuild en getests. Er kan ook worden gereleased, maar dit is niet meegenomen in de pipeline. Hieronder een historie van verschillende jobs (build & tests) met de pipeline die we hebben opgezet.

AircraftTracker-ASP.NET Core (.NET ...

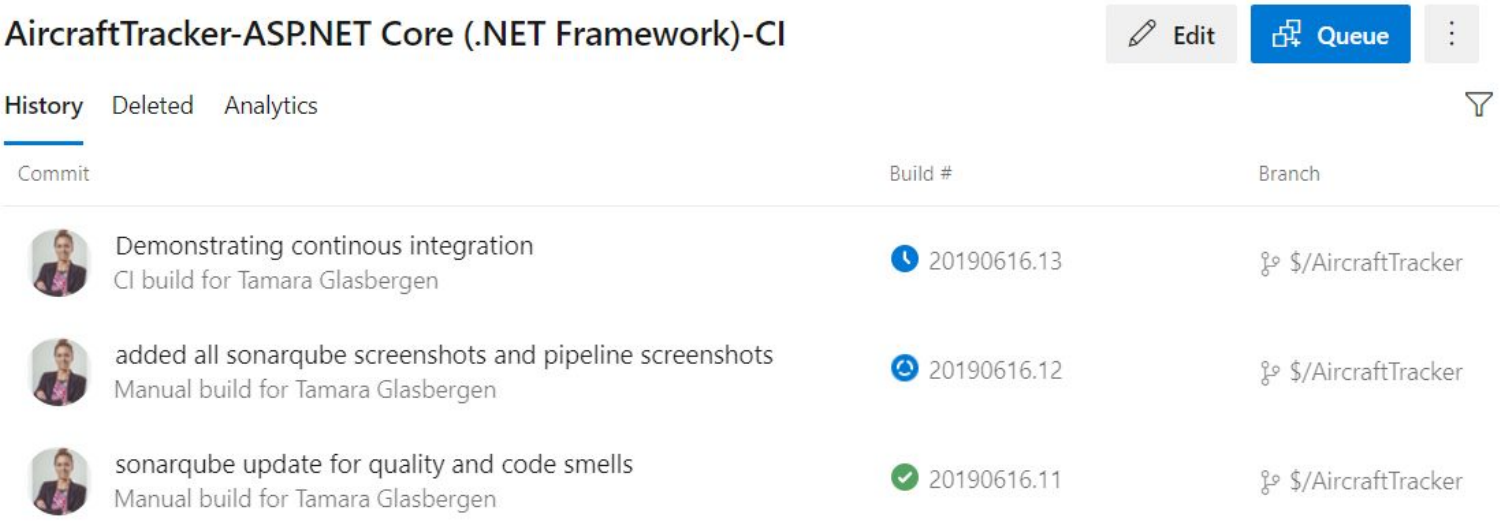
Tasks Variables Triggers Options Retention History | Save & queue Discard Summary Queue ...

Changed By	Change Type	Changed Date	Comment
Tamara Glasbergen	Update	6/16/2019 6:43 PM	
Tamara Glasbergen	Update	6/16/2019 4:32 PM	
Tamara Glasbergen	Update	6/16/2019 4:24 PM	
Tamara Glasbergen	Update	6/16/2019 3:57 PM	
Tamara Glasbergen	Update	6/16/2019 3:42 PM	
Tamara Glasbergen	Update	6/16/2019 3:13 PM	
Tamara Glasbergen	Update	6/16/2019 2:39 PM	
Tamara Glasbergen	Add	5/19/2019 2:30 PM	

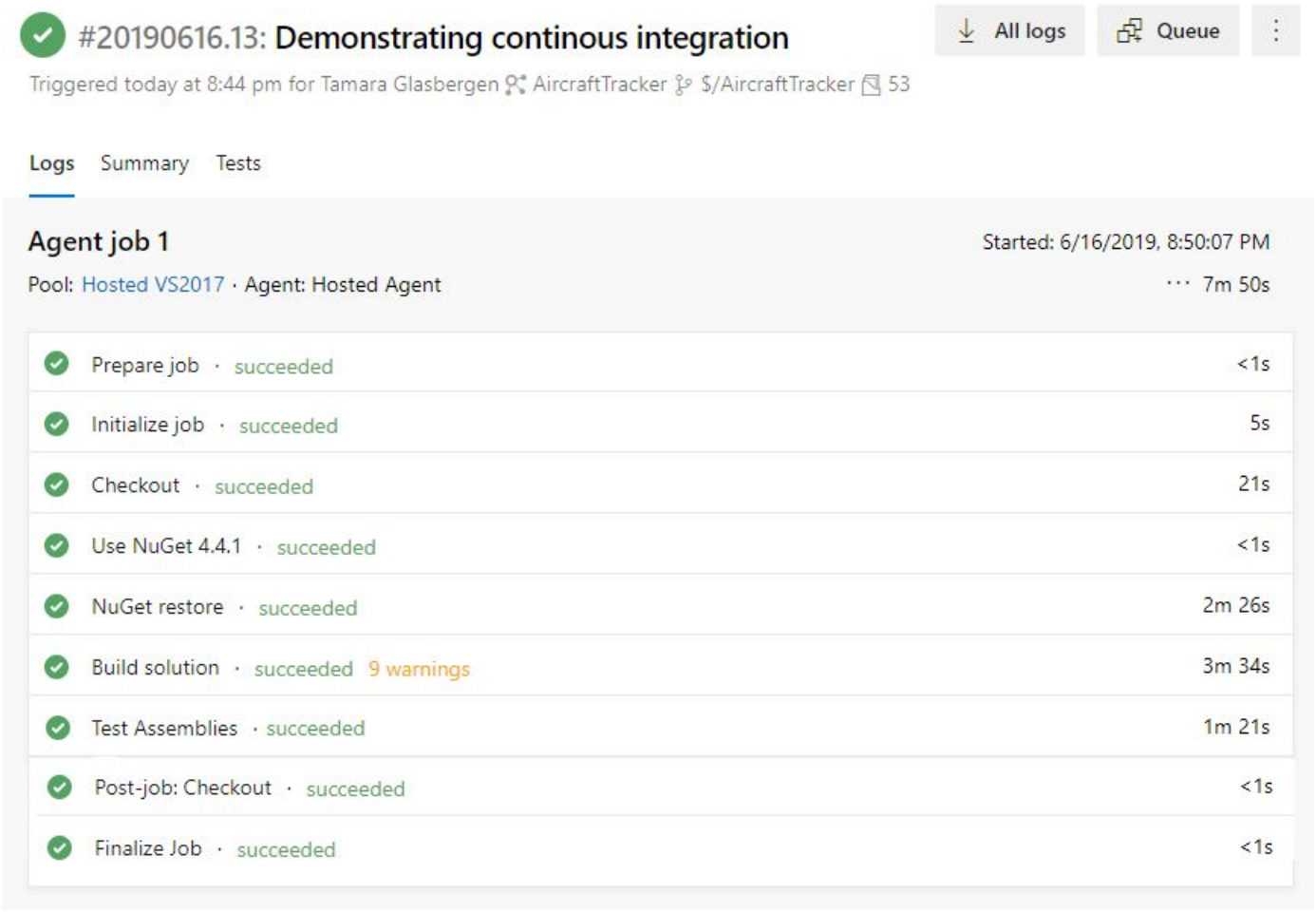
Tijdens zo'n job worden een aantal stappen doorlopen die allemaal geconfigureerd zijn voor onze projecten. Hieronder een klein stukje van de output window tijdens een build van een job:

```
    Adding file (d:\a\3\s\SkyWebApp\SkyWebApp\obj\Release\netcoreapp2.2\PubTmp\Out\wwwroot\dist\toolkit-inverse.css).
    Adding file (d:\a\3\s\SkyWebApp\SkyWebApp\obj\Release\netcoreapp2.2\PubTmp\Out\wwwroot\dist\vendor-manifest.json).
    Adding file (d:\a\3\s\SkyWebApp\SkyWebApp\obj\Release\netcoreapp2.2\PubTmp\Out\wwwroot\dist\vendor.css).
    Adding file (d:\a\3\s\SkyWebApp\SkyWebApp\obj\Release\netcoreapp2.2\PubTmp\Out\wwwroot\dist\vendor.js).
    Updating declared parameter 'IIS Web Application Name'.
    Successfully executed Web deployment task.
    Package "WebApp.zip" is successfully created as single file at the following location:
    file:///d:/a/3/a
    To get the instructions on how to deploy the web package please visit the following link:
    http://go.microsoft.com/fwlink/?LinkId=124618
  Done Building Project "d:\a\3\s\SkyWebApp\SkyWebApp\SkyWebApp.csproj" (default targets).
  Done Building Project "d:\a\3\s\SkyWebApp\SkyWebApp.sln" (default targets).

  Build succeeded.
      0 Warning(s)
      0 Error(s)

  Time Elapsed 00:01:44.23
  ##[section]Finishing: Build solution
```

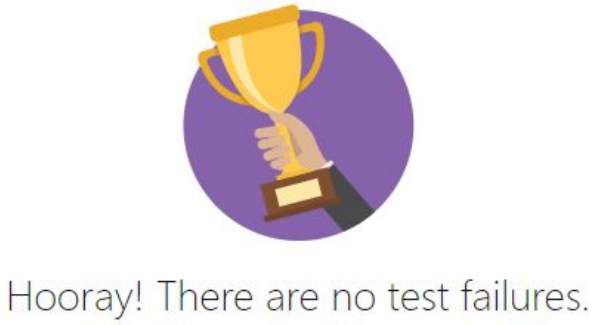
Het is mogelijk om zowel een manual job te runnen, als een job te triggeren, bijvoorbeeld door een check-in. Hieronder een screenshot van een al voltooide job, een manual job die bezig is met voltooien, en een getriggerde job die in de wachtrij staat om te starten.



In zo'n pipeline worden er verschillende dingen gedaan. Deze worden uitgebeeld in de afbeelding hieronder. Hier zie je een triggered build door een incheck die onder de naam 'Demonstrating continuous integration' is ingecheckt. Deze heeft eerst de NuGet packages geïnstalleerd, daarna de verschillende solutions gebuild, daarna de tests gedraaid die hieraan verbonden zijn om vervolgens de job weer af te sluiten.



De job heeft in dit geval 7 minuten en 50 seconden nodig om te voltooien. Dit heeft voornamelijk te maken met het installeren van de NuGet Packages en tijdens de build het wachten op de webpack build, wat een lange tijd duurt. Al met al is dit een redelijke snelheid, waar we zeer tevreden mee zijn de eerste keer dat we zoiets opzetten.



De testresultaten kun je los bekijken zoals duidelijk wordt in het plaatje hierboven. In de afgelopen 14 dagen is er 1 test is geweest die heeft gefaald. Nu slagen alle tests dus dat is in de tussentijd nog opgelost waardoor de pipeline job is geslaagd.

Reflectie

De matrix waarmee je de beste manier kan vinden om user requirements boven water te krijgen vinden we, vooral Sirée, een zeer interessant instrument. Tot nu toe hebben we altijd naar de techniek gegrepen die ons het meest bekend was (namelijk het interviewen van de opdrachtgever). Maar wanneer de opdrachtgever vakkundig is als programmeur, dan is het handiger om standaard vragenlijsten te gaan gebruiken. Je vergeet dan geen details uit te vragen en de opdrachtgever snapt toch wel wat er gevraagd wordt. We hadden dit graag wat eerder in het semester geweten, want de vragenlijsten nasturen kwam als mosterd na de maaltijd.

Tamara geeft aan dat ze vooral Sonarqube heel erg waardeert. De directe feedback op dat je code niet voldoet, zorgt ervoor dat je ter plekke gelijk al gaat refactoren totdat je geen smells of debt hebt. Je belandt niet in een discussie met anderen over of het beter is of niet. De computer says so en dus ga je aan het werk. Siree geeft aan dat ze vooral het TDD waardeert. Het dwingt je van te voren na te denken over de code die je schrijft. Je snapt daardoor conceptueel beter wat je doet en raakt niet het overzicht kwijt van wat je probeert te bereiken terwijl je je code werkend probeert te krijgen.

We zagen beide niet gelijk de voordelen van de pipeline in. Het kostte veel tijd om het op te zetten en gezien de omstandigheden van Sky Reckon (nog geen klanten), heeft het ook geen zin om een continuous delivery-methode op te zetten. Daarnaast heeft Siree ook het algemene bezwaar dat als er een grotere change wordt doorgevoerd, de eindgebruikers daarvan van te voren op de hoogte moeten worden gebracht. Dit gaat onder zeer grote tijdsdruk bij continuous delivery. Binnen een organisatie als de gemeente Amsterdam zag zij dit dan ook niet als toepasbaar.

Appendix A

Questionnaires SkyReckon for requirements.

#	Questions	Supported	Risk	Design Decisions and Location	Rationale and Assumptions
1	Does the system use ping/echo to detect a failure of a component or connection, or network congestion?	No	L	Het is mogelijk doordat alle services los van elkaar staan, maar is op dit moment niet ingeregeld	Infrastructuur staat nog niet 100% vast. Moment dat dit stabiel wordt worden dit soort dingen belangrijk. Impact is laag, risico dat dit voorkomt is laag.
2	Does the system use a component to monitor the state of health of other parts of the system?	No	L	Hiervoor staat een aparte service klaar voor de SaaS applicatie, maar is nog niet in gebruik. Wel is Logstash en Kubernetes in gebruik waar standaard monitoring in zit voor DevOps.	Infrastructuur staat nog niet 100% vast. Moment dat dit stabiel wordt worden dit soort dingen belangrijk. Impact is laag, risico dat dit voorkomt is laag.
3	Does the system use a heartbeat to detect a failure of a component or connection or network congestion?	Yes	M	De consumers versturen elke 10 minuten een message dat deze nog alive zijn. Andere services hebben geen heartbeat.	Consumers zijn vanaf buiten niet te bereiken en daarvoor is er een heartbeat ingeregeld om te laten weten dat ze nog alive zijn. Andere services is niet nodig. Kubernetes regelt de servers en de webapi's en andere services kunnen in de toekomst met een ping gecheckt worden.
4	Does the system use a timestamp to detect incorrect sequences of events in distributed systems?	Yes	M	Alle componenten hebben een timestamp, maar niet alles wordt gebruikt om automatisch te checken op fouten. De data in de back-end hebben meta-data, daarnaast wordt	meta-data wordt op dit moment niet gevalideerd. In een later stadium wordt dit uitgebreid met checks op timestamps. Omdat events niet vluchtig zijn kan op een later moment ook nog

				logstash gebruikt waar altijd een timestamp gebruikt wordt.	uitgebreidere validaties/conversies gedaan worden.
5	Does the system do any sanity checking: checking the validity or reasonableness of a component's operations or outputs?	No	H	Op dit moment is er geen uitgebreide zelftest voor het systeem.	Dit is in dit stadium van productie ook nog niet aan de orde.
6	Does the system do condition monitoring, checking conditions in a process or device, or validating assumptions made during the design?	Yes	H	Er worden in alle lagen van de gehele applicatie gecheckt op input of de input voldoet aan de verwachte input. Ook tussen compile-time and run-time van Typescript zitten checks.	Een belangrijke om er voor te zorgen dat in productie de applicatie niet failed.
7	Does the system use voting to check that replicated components are producing the same results?	No	L	De aanname is dat altijd hetzelfde gebeurd, dus op dit moment is dit overhead.	Er is op dit moment is er geen berekening die dusdanig belangrijk is dat het risico te groot wordt om hierop te checken. Er kan binnen het systeem gecorrigeerd worden.
8	Do you use exception detection to detect a system condition that alters the normal flow or exception?	Yes	L	Kubernetes heeft altijd een desired state, het systeem wordt gemonitord en bij afwijking wordt een corrigerende actie uitgevoerd. Ook de consumers houden een verbinding open en monitoren of deze bereikbaar blijft	Andere services kunnen door middel van ping/echo gecheckt worden en het is niet nodig om op meerdere lagen exception detection te plaatsen
9	Can the system do a self-test to test itself for correct operation?	No	M	Er wordt niet getest of het systeem operable is door middel van een zelftest.	Een tool gaat onder een testgebruiker-account checken of het systeem goed functioneert en alle onderdelen met elkaar samenwerken.

10	Does the system employ active redundancy?	No	L	In de consumers zou active redundancy kunnen voorkomen, in de rest van het systeem niet	Het moment dat er zowel in productie als in development dezelfde consumer wordt aangeroepen zouden er twee keer data opgeslagen kunnen worden wat het ander overschrijft. Principe LIFO wordt gehanteerd.
11	Does the system employ passive redundancy?	Yes	H	Kubernetes ondersteunt clusters en houdt hier actief meerdere kopieën bij van de data. De back-end, Kafka, draait in meerdere partities.	Meerdere clusters zijn in de toekomst een mogelijkheid. De standaardconfiguratie van Kubernetes is op dit moment voldoende. Kafka in Kubernetes met meerdere partities ook.
12	Does the system employ spares?	No	H	Het moment dat er daadwerkelijk in productie gedraaid wordt, word dit belangrijk en heeft het een grote impact.	Het idee is dat de staging environment ook gebruikt gaat worden voor back-up, waar de klanten in production environment handelen.
13	Does the system employ exception handling to deal with faults?	Yes	H	In alle lagen is exception handling in gebruik	De klanten mogen niet in foutstatus terecht komen waardoor zij niks meer met de applicatie kunnen doen.
14	Does the system employ rollback, so that it can revert to a previously saved good state in the event of a fault?	Yes	H	Het is belangrijk om altijd een audit te hebben van elk object. Er wordt geen data weggegooid en er is altijd een rollback te doen.	Een volledige audit trail van data is een vereiste, daarnaast met updates is het de bedoeling dat er altijd een werkende applicatie is.
15	Can the system perform in-service software upgrades to executable code images in a non-service-affectin g manner?	Yes	H	Kubernetes kan verschillende versies naast elkaar draaien en de load-balance (traffic) overzetten naar een nieuwe versie zonder dat er downtime is	Om continous integration te faciliteren is het nodig om verschillende versies naast elkaar te kunnen draaien, dit geldt voor alle onderdelen van de applicatie. Klanten

					willen niks weten van downtime door updates.
16	Does the system systematically retry in cases where the component or connection failure may be transient?	Yes	H	Alle connecties tussen componenten hebben systematische retry's. Kubernetes probeert componenten uit failure te krijgen door te herstarten.	Connecties hebben een retry, omdat softwarecomponenten herstart kunnen worden in de achtergrond en weer online kunnen komen.
17	Can the system simply ignore faulty behavior?	Yes	M	Garbage in, garbage out in data. Doordat er met microservices gewerkt wordt kan failure in een van deze services totaal genegeerd worden door andere services	In andere lagen van de applicatie wil je altijd weten als er fouten gebeuren, daardoor is het niet gewenst om in andere lagen dit gedrag te negeren, maar dient er altijd een melding of actie aan verbonden te zitten.
18	Does the system have a policy of degradation when resources are compromised, maintaining the most critical system functions in the presence of component failures, and dropping less critical functions?	No	L	Wordt niet geambieerd.	Inrichting van degradation creëert teveel overhead. Daarnaast is er genoeg inzicht in gebruik om bijvoorbeeld memory load te managen.
19	Does the system have consistent policies and mechanism for reconfiguration after failures, re-assigning responsibilities to the resources left functioning, while maintaining as much functionality as possible?	Yes	H	Kubernetes regelt automatisch herconfiguratie op het moment dat er delen van het systeem uitvallen aan de hand van de desired state	Het automatisch laten uitvoeren van herconfiguratie is sneller, betrouwbaarder en bespaard kosten in vergelijking met het handmatig opnieuw configureren van het systeem.

20	Can the system operate a previously failed or in-service upgraded component in a “shadow mode” for a predefined time prior to reverting the component back to an active role?	No	L	Shadow mode is niet nodig	Omdat de staging environment al in de productieomgeving gedraaid wordt, heb je al een soort shadow mode.
21	If the system uses active or passive redundancy, does it also employ state resynchronization, to send state information from active to standby components?	Yes	M	Kafka nodes communiceren met elkaar en synchroniseren automatisch na standby	Andere clusters zijn gebaseerd op Kafka
22	Does the system employ escalating restart - that is, does it recover from faults by varying the granularity of the components restarted and minimizing the level of service affected?	Yes	H	Het moment dat er een service uit ligt wordt deze automatisch opnieuw opgestart door Kubernetes. Binnen de software zitten er exceptions op maar geen automatic restart after failure	Kubernetes is een efficiënte manier om falende services te managen. Binnen de software worden er andere technieken gebruikt om failure te voorkomen.
23	Can the system remove components from service, temporarily placing a system component in an out-of-service state, for purpose of mitigating potential system failures?	Yes	L	Ja dit is mogelijk doordat er gewerkt wordt met microservices, maar dit is niet wenselijk.	Er kunnen meerdere versies van hetzelfde component naast elkaar gedraaid worden waardoor out of service niet perse nodig is. Traffic kan gecontroleerd overgezet worden naar de nieuwe versie.
24	Does the system employ transactions - bundling state	Yes	H	Kafka zit tussen de communicatie van alle componenten in, slaat alle communicatie op	Door het gebruik van Kafka is er een single source of truth

	updates so that asynchronous messages exchanged between distributed components are atomic, consistent, isolated and durable?			en zorgt voor aflevering en mogelijk herhaling van de message.	
25	Does the system use a predictive model to monitor the state of health of a component to ensure that the system is operating within nominal parameters?	No	L	Op dit moment niet nodig	In productie is het mogelijk om bijvoorbeeld via ELK-stack de state te voorspellen. Dit speelt op dit moment niet vanwege het lage aantal klanten
26	Does the system prevent exceptions from occurring by, for example, masking a fault, using smart pointers, abstract data types, or wrappers?	Yes	H	Door het gebruik van interfaces en data typing in Typescript en C# worden excepties voorkomen	Fouten worden eerder afgevangen en gebruikers kunnen eerder feedback ontvangen over juiste data
27	Has the system been designed to increase its competence set, for example by designing a component to handle more cases/faults - as part of its normal operation?	Yes	H	Alles is schaalbaar. Clusters zorgen voor juiste distributie, meerdere services kunnen naast elkaar gedraaid.	In een groeiende markt is het nodig om een systeem te hebben wat schaalbaar is, en dus zijn alle componenten hierop uitgekozen.