Professionele Bachelor Toegepaste Informatica

Selecteren en visualiseren van software-kwaliteitswaarden

Khena Jacobs

Promotoren:

Timo Naessens, Jonny Daenen — Selligent Marketing Cloud
Lowie Vangaal — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Professionele Bachelor Toegepaste Informatica

Selecteren en visualiseren van software-kwaliteitswaarden

Khena Jacobs

Promotoren:

Timo Naessens, Jonny Daenen — Selligent Marketing Cloud
Lowie Vangaal — Hogeschool PXL Hasselt

Bachelorpaper Academiejaar 2018-2019

Dankwoord

Als student toegepaste informatica bij PXL kreeg ik de kans om een stageopdracht die voor mij een grote uitdaging was om met succes uit te voeren. De 12 intensieve maar vooral leerrijke weken waren voor mij een unieke kans om mijzelf uit mijn *comfort zone* te plaatsen en nieuwe grenzen in mijn vaardigheden te ontdekken.

Dit eindwerk was nooit zo compleet en diepgaand geweest zonder de hulp, steun en expertise van alle personen die mij geholpen hebben tijdens mijn stageperiode bij Selligent.

Ik wil graag mijn stagepromoter Lowie Vangaal bedanken om op constante basis mij feedback te geven op mijn stage en eindwerk. Ik begrijp dat het zeer tijdrovend is om voor elke student waar hij verantwoordelijk voor is de nodige hulp aan te bieden. Ik ben hem dan ook erg dankbaar voor de tijd en moeite die hij heeft geïnvesteerd in mij.

In het bijzonder wil ik Timo Naessens en Jonny Daenen bedanken voor de opportuniteiten die ik kreeg om nieuwe domeinen te verkennen en mij de kans te geven om extra inzichten te krijgen binnen een internationaal bedrijf. Ik ben Timo en Jonny heel erg dankbaar voor de tijd en moeite die ze gestoken hebben in het opvolgen van al mijn documentatie en de uitgebreide feedback die ik kreeg om mijn stageperiode zo compleet en correct mogelijk te maken. Het feit dat beiden beslist hebben om niet in het onderwijs te staan en zoveel moeite in een student willen steken, betekent heel erg veel voor mij.

Ik wil ook hogeschool PXL bedanken voor alle kansen en ervaringen die ik doorheen mijn carrière als student kreeg. Ik kan alleen maar alle andere studenten dezelfde kansen toewensen.

Khena Jacobs

Abstract

Bij het ontwikkelen van software is de kwaliteit uitermate belangrijk. Indien die ontoereikend wordt, heeft dat onmiddellijk een nadelig effect op de gebruiker van de software. Daarom is een continue opvolging hiervan noodzakelijk.

In deze stage wordt de betekenis van softwarekwaliteit onderzocht. Er wordt gezocht naar goede meetwaarden en de bijbehorende data. Ten slotte wordt een betekenisvol *dashboard* gebouwd dat de *Software-engineers* en *product managers* in staat stelt om de kwaliteit op te volgen. Allereerst wordt onderzocht wat software-kwaliteit precies is. Er wordt bekeken hoe kwaliteit wordt gedefinieerd en gemeten, alsook in welke categorieën die meetwaarden opgedeeld worden (bv. Product en proces). Ook wordt bekeken op welke manier kwaliteit de performantie van bedrijven kan beïnvloeden.

Vervolgens wordt nagegaan welke van deze meetwaarden interessant zijn voor Selligent. Daarnaast wordt geïdentificeerd op welke locaties de brondata bewaard worden. Dat houdt in dat er voor elke interessante meetwaarde aangegeven wordt waar deze data zich bevindt indien deze aanwezig zijn (welke service/server, waar in het proces, etc.). Hierna worden de meetwaarden gevisualiseerd met behulp van Power BI. Er wordt beknopt een vergelijking gedaan met Python *tool* Plot.ly en *library* Pandas. Er wordt op zoek gegaan naar de best mogelijke visualisatie voor de verschillende meetwaarden. Vervolgens worden de beschikbare waarden in een interactief *dashboard* geplaatst om zo in één oogopslag een algemeen overzicht te geven van de kwaliteit over de tijd heen.

De stage en het onderzoek worden uitgevoerd bij het internationale bedrijf Selligent, rond het product “Selligent Marketing Cloud”. Dat is een marketing automatisatie tool die het leven van *marketeers* binnen grote bedrijven (kranten, media, *retail*-ketens) eenvoudiger maakt. De tool laat toe om op basis van profieldata en gebruikersgedrag een persoonlijke marketingboodschap te zenden via verschillende kanalen (e-mail, mobiele applicaties, etc.). Vervolgens wordt de effectiviteit hiervan opgevolgd. Het resultaat van de stage wordt indien mogelijk rechtstreeks geïntegreerd in de processen en in de interne *cloud*-gebaseerde *dashboarding* pipeline.

Inhoudsopgave

Inhoudsopgave

Dankwoord ii

Abstract iii

Inhoudsopgave iv

Lijst van gebruikte figuren ix

Lijst van gebruikte tabellen xi

Lijst van gebruikte termen xii

Inleiding 1

I. Stageverslag 2

1 Bedrijfsvoorstelling 2

2 Voorstelling Stageopdracht 3

2.1 Situering stageopdracht 3

2.1 Inleiding stageopdracht 3

2.2 Uitwerking stageopdracht 3

3 Bedrijfshiërarchie 5

4 Procespipeline 6

5 Databronnen 7

5.1 Inleiding 7

5.2 Azure DevOps 7

5.2.1 Introductie 7

5.2.2 Databeschikbaarheid 7

5.2.3 Data verzamelen uit Azure DevOps 8

5.3 Data verzamelen uit Robot Framework 12

5.4 Toolpipeline 13

6 Visualisatietools 15

6.1 Inleiding 15

6.2 Toolkeuze 15

6.3 Jupyter Notebook 15

6.3.1 Introductie 15

6.3.2 Instapniveau 15

6.3.3 Voorbereiding 16

6.3.4 Data Cleaning 16

6.3.5 Update Datasets 18

6.3.6 Live Data 18

6.3.7 Visualiseren van waarden 18
6.3.8 Gebruik van Plot.ly en Pandas 20
6.3.9 Visualisatiemogelijkheden 22
6.3.10 Opbouw Dashboard 22
6.4 Power BI 23
6.4.1 Introductie 23
6.4.2 Instapniveau 23
6.4.3 Voorbereiding 23
6.4.4 Data Cleaning 23
6.4.5 Updaten Datasets 24
6.4.6 Visualisatiemogelijkheden 25
6.4.7 Live data visualiseren 25
6.4.8 Opbouw Dashboard 25
6.5 Azure DevOps 26
6.5.1 Introductie 26
6.5.2 Instapniveau 26
6.5.3 Voorbereiding 26
6.5.4 Data Cleaning 26
6.5.5 Updaten Datasets 27
6.5.6 Live data 28
6.5.7 Gebruik van Azure DevOps 28
6.5.8 Visualisatiemogelijkheden 29
6.5.9 Opbouw Dashboard 29
6.6 Toolvergelijking 30
7 Visualisaties 32
7.1 Kwaliteitsdashboard 32
7.1.1 Inleiding 32
7.1.2 Databron Diagram 32
7.1.3 Mockup 33
7.1.4 Uitwerking kwaliteitsdashboard 36
7.1.5 Bug gerelateerd aan Pink Tickets 37
7.1.6 Releasewaarden 38
7.1.7 Bugs aanwezig in de Software 39
7.1.8 Test Pyramide en resultaten Robot Framework 40
7.1.9 Interpretatie van het dashboard 41
7.1.10 Verdere optimalisatie 41

7.1.11 Conclusie productkwaliteit 42
7.2 Team Dashboard 42
7.2.1 Inleiding 42
7.2.2 Databron diagram 42
7.2.3 Uitwerking Dashboard 43
7.2.4 Interpretatie van het dashboard 47
7.2.5 Conclusie proceskwaliteit 47
7.2.6 Verdere optimalisatie 48
7.3 Productie dashboard 49
7.3.1 Inleiding 49
7.3.2 Databron diagram 49
7.3.3 Uitwerking dashboard 49
7.4 Rapport automatische testen 51
7.4.1 Inleiding 51
7.4.2 Databron Diagram 51
7.4.3 Opbouw Rapport 51
7.5 Sales Dashboard 53
7.5.1 Inleiding 53
7.5.2 Mockup 53
II. Onderzoekstopic 54
1 Probleemstelling 54
2 Onderzoeksmethode 55
3 Kwaliteitswaarden 56
3.1 Literatuurstudie 56
3.2 Inleiding 56
3.3 Verzamelen van waarden 56
3.4 Product Development 57
3.4.1 Levensduur van de workbranch: [7] 57
3.4.2 Actieve branches per team [7] 57
3.4.3 Status van pull request [7] 57
3.4.4 Tijd gespendeerd aan productvoorbereiding [7] 58
3.4.5 Tijd gespendeerd aan product development [14][7] 58
3.4.6 Deployment Frequentie [14][7] 58
3.4.7 Deployment Fail Ratio [7] 59
3.4.8 Gedupliceerde code [3] 59
3.4.9 Code Smells [3] 59

3.5 Bugs 60
3.5.1 Bug Density [1] 60
3.5.2 Lead Time van Bugs [7][11] 60
3.5.3 Cycle Time van Bugs [7][11] 61
3.5.4 Werkballast van Bugs 61
3.5.5 Staat van Bugs [11] 62
3.5.6 Ernst van Bugs 63
3.5.7 Escape Rate van Bugs [1][12] 63
3.5.8 Bugs gevonden tijdens Unit Testen [7][13] 64
3.5.9 Bugs gevonden tijdens Smoke testen [13][14] 64
3.5.10 Bugs gevonden tijdens Regression testen [7][13][14] 64
3.5.11 Bugs gevonden door de eindgebruikers 64
3.6 Features 65
3.6.1 Staat van Features 65
3.7 Sprints 66
3.7.1 Status van Work Items 66
3.7.2 Sprintgrootte 67
3.7.3 Vooruitgang in werk [11] 67
3.7.4 Team Tevredenheid [7] 68
3.8 Testfasen 69
3.8.1 Aantal test cases: Unit testen [13] 69
3.8.2 Code Coverage [7][13] 69
3.8.3 Resultaten: Smoke testen [13][14] 69
3.8.4 Resultaten: Regression Testen [13][14] 70
3.8.5 Manuele en automatische testen [7] 70
3.9 Klantenfeedback 71
3.9.1 Lead time van pink tickets [7][11] 71
3.9.2 Cycle Time van Customer Feedback [7][11] 72
3.9.3 Status van pink tickets 73
3.9.4 Ernst van pink tickets 73
3.9.5 Klanttevredenheid [7] 74
3.10 Technical *Debt* 75
3.10.1 Hoeveelheid Technical Debt 75
3.10.2 Status van Technical Debt [3] 75
3.10.3 Ernst van Technical Debt 77
3.11 Werkkwaliteit 78

3.11.1 Tijd gespendeerd aan proactief werk [17] 78
3.11.2 Tijd gespendeerd aan reactief werk [17] 78
3.12 Wat is Softwarekwaliteit? 78
3.13 State of DevOps 78
3.14 Conclusie 79
4 Visualisatiemogelijkheden 80
4.1 Inleiding 80
4.2 Historische Data 80
4.3 Vergelijken en Rangschikken 81
4.4 Conclusie 83
III. Conclusie 84
1 Meetbare waarden softwarekwaliteit 84
2 Selectie voor meten product- en proceskwaliteit 85
2.1 Productkwaliteit 85
2.2 Proceskwaliteit 86
3 Blootleggen van anomalieën 86
4 Vergelijking visualisatietools 87
5 Status van Selligent productkwaliteit 88
6 Status van Selligent proceskwaliteit 88
Bibliografie 89
Bijlagen 91

Lijst van gebruikte figuren

Figuur 1:Organigram Selligent. 5
Figuur 2: Procespipeline aangevuld met testtools die gebruikt worden binnen Selligent. 6
Figuur 3:Schematische voorstelling toolpipeline. 7
Figuur 4: Query scherm in Azure DevOps. (Dummy data) 8
Figuur 5: Output query items met behulp van API. 9
Figuur 6: Automatische output Power BI van .txt-bestand. 10
Figuur 7: Invullen van voorwaarden in Analytics View. 11
Figuur 8: Verificatie voor Power BI. 12
Figuur 9: Selectie van Azure DevOps als databron. 12
Figuur 10: Output van automatische test cases van Robot Framework. 13
Figuur 11:Schematische voorstelling toolgebruik en ophalen van data. 14
Figuur 12: Visualisatiemogelijkheden Plot.ly. 22
Figuur 13: Edit functie Azure DevOps. 27
Figuur 14: Bewerken van geselecteerde waarden. 27
Figuur 15:Toevoegen van kolommen in query. 28
Figuur 16:Visualisatiemogelijkheden in Azure DevOps. 29
Figuur 17: Databron diagram van het kwaliteitsdashboard. 32
Figuur 18: Mockup dashboard softwarekwaliteit. (Dummy Data) 33
Figuur 19: Bugs gelinkt aan pink tickets. (Dummy Data) 34
Figuur 20: Onderdeel bugs en hotfixen (links) en bugs gesorteerd per ernst (rechts). (Dummy Data) 35
Figuur 21: Tevredenheid van laatste release. (Dummy Data) 35
Figuur 22: Kwaliteitsdashboard uitgewerkt in Power BI. 36
Figuur 23: Bugs gelinkt aan pink tickets. 37
Figuur 24: Releasewaarden. 38
Figuur 25: Aantal bugs actief doorheen de tijd. 39
Figuur 26:Testpyramide (Links) en resultaten Robot Framework (rechts). 40
Figuur 28:Databrondiagram van het teamdashboard. 42
Figuur 29: Sprint overzicht van het team dashboard voor één team. 43
Figuur 30: Backlog overzicht van het team dashboard voor één team. 44
Figuur 31: Area overzicht van het team dashboard voor één team. 45
Figuur 32: Performance overzicht van het team dashboard voor één team. 46
Figuur 33:Databron Productiedashboard. 49
Figuur 34: Dashboard priority lijst. 49
Figuur 35: Dashboard bugs. 50
Figuur 36: Dashboard manuele test cases. 50
Figuur 37: Databron diagram van Rapport automatische testen. 51
Figuur 38: Voorbeeld Test Case. 51
Figuur 39: Bepalen van 'true flaky' test cases. 52
Figuur 40: Output in Jupyter Notebook van 'True Flaky' test cases. 52
Figuur 41: Mockup Salesdashboard. (Dummy Data) 53
Figuur 42: Onderverdeling kwaliteitswaarden. 56
Figuur 43: Lead time van bugs in 2017, 2018, 2019. 60
Figuur 44: Vergelijking lead, reaction en cycle time. [15] 61
Figuur 45: Cycle time van bugs in 2017, 2018 en 2019. 61
Figuur 46: Werkballast van bugs aanwezig in de backlog. 62
Figuur 47: Statussen van bugs aanwezig in de backlog. 62

Figuur 48: Ernst van bugs aanwezig in backlog (links) en ernst vergeleken per sprint (rechts). 63
Figuur 49: Features per sprint (links) en alle features aanwezig in de backlog (rechts). 65
Figuur 50: Statussen van work items in de backlog (links) en vergeleken per sprint (rechts). 66
Figuur 51: Afgewerkte story points per sprint. 67
Figuur 52: Work items gesorteerd per staat (links) en per staat weergegeven per sprint (rechts)..... 68
Figuur 53: Pyramidevorm per testfase. 70
Figuur 54: Aantal pink tickets voor actieve sprint (links) en per sprint (rechts). 71
Figuur 55: Lead time van pink tickets. 72
Figuur 56: Cycle time van pink tickets. 72
Figuur 57: Status van pink tickets in actieve sprint (links) en als overzicht per sprint (rechts). 73
Figuur 58: Ernst van pink tickets in actieve sprint (links) en per sprint (rechts). 74
Figuur 59: Fictieve visualisatie klant- en teamtevredenheid. (Dummy Data) 74
Figuur 60: Hoeveelheid actieve technical debt opgenomen per sprint. 75
Figuur 61: Staat van technical debt voor huidige sprint (links) en per sprint (rechts). 76
Figuur 62: Ernst van technical debt per sprint. 77
Figuur 63: Area plot voorbeeld 80
Figuur 64: Voorbeeld van segmentatie. 82
Figuur 65: Onderverdeling kwaliteitswaarden 84
Figuur 66: Selectie kwaliteitswaarden productkwaliteit 85
Figuur 67: Selectie kwaliteitswaarden proceskwaliteit 86

Lijst van gebruikte tabellen

Tabel 1: Toolvergelijking..... 30

Lijst van gebruikte termen

Analytics view	Methode binnen Azure DevOps om data te selecteren en te verwerken in een tabel.
Azure DevOps	Azure DevOps is een developer service van Microsoft die gebruikt wordt om teams te helpen bij het inplannen van werk, samenwerken in code development en het bouwen, testplannen opstellen en vele andere mogelijkheden
Backlog	Lijst van onderdelen die gelinkt zijn aan de productie van een applicatie.
Build	Een klein onderdeel van een applicatie waaraan gewerkt is.
Continuous Delivery	Een methode binnen softwareontwikkeling die streeft naar een applicatie die te allen tijde bruikbaar is voor de klant.
Cycle time	De tijd die verstreken is om een work item af te werken.
Dashboard	Een scherm met visuele aspecten als grafieken en diagrammen die gelinkt zijn aan data.
Deployment	De uitvoering van een afgewerkt item naar de pipeline.
Epic	Een groter geheel van een applicatie dat ontwikkeld moet worden. Een epic wordt verdeeld in kleinere onderdelen (features) om eenvoudiger doelen te stellen tijdens softwareproductie.
Escape Rate	De ratio aan bugs die terugkomen ondanks deze als opgelost gemeld staan in de backlog.
Features	Grotere delen van een programma. Vaak zijn die functionaliteiten die in kleinere delen opgesplitst zijn om de werkverdeling efficiënter te laten lopen.
Lead time	De tijd die gespendeerd wordt aan een item zodra het aanwezig is de backlog tot deze afgewerkt is.
Merge	Een samenvoeging van meerdere onderdelen in de branch van de ontwikkelde software.
Pandas	Python library die gebruikt wordt om data te verwerken, bewerken en te visualiseren met behulp van grafieken en tabellen.
Pink ticket	Een support item dat gelogd wordt door klanten.
Pipeline	Een proces dat opgedeeld is in verschillende fasen en dat de rode draad is om aan softwareontwikkeling te doen.
Plot.ly	Een Python applicatie die gebruikt wordt om aan data visualisatie te doen op een interactieve wijze.
Power BI	Een Microsoftapplicatie die dient om aan data te bewerken, analyseren, verwerken en op een interactieve manier te visualiseren.
Pull request	Aanvraag die gedaan wordt wanneer een stuk code of onderdeel van de applicatie afgewerkt is en deze in de mainbranch toe te voegen.
Regression Testen	Testfase binnen softwareontwikkeling waarin testen uitgevoerd worden om te controleren of bepaalde onderdelen nog werken na er een wijziging doorgevoerd is.
Release	Na het succesvol doorlopen van de productiepipeline krijgt de software een nieuwe release versie.
Repository	Opslagplaats van de ontwikkelde software waar alle code aanwezig is met de specifieke branches.
Requirements	Opgestelde eisen waaraan software moet aan voldoen. Dat zijn eisen die in samenspraak zijn met wat klanten willen.
Smoke testen	Test fase binnen softwareontwikkeling waarin de belangrijkste functies getest worden van de software.

SonarQube	Code analyse tool die gebruikt wordt om de code kwaliteit de beoordelen van de gebouwde software.
Sprint	Een duur van twee werkweken waarin werk in een lijst wordt bijgehouden om af te werken tijdens de sprint.
Story points	Punten die gelinkt worden aan de hoeveelheid werk en tijd er geïnvesteerd moet worden in specifieke items in de backlog.
Suite	Onderverdeling om test *cases* in bij te houden en makkelijker op te vragen.
Tag	Toevoeging aan items in de backlog om eenvoudig te kunnen sorteren op basis van de naam die gegeven wordt als tag.
Technical debt	Problemen die opgelost zijn voor een kortere termijn maar over een langere periode opnieuw een probleem kunnen worden.
Unit testen	Test fase tijdens de softwareontwikkeling die nagaat of er bugs en niet werkende code aanwezig zijn in de gebouwde onderdelen.
Visualisatietool	Tool die dient om data te visualiseren om conclusies te kunnen trekken uit de resultaten.
Work items of items	Een onderwerp dat in de backlog bijgehouden wordt als werk dat verricht moet worden tijdens softwareontwikkeling.
Workbranch of branch	Onderdeel van de software waarin code opgeslagen wordt van specifieke functionaliteiten die later in een mainbranch gestoken worden. De mainbranch is het geheel van alle kleinere branches samengevoegd.
QA	Quality Assurance: De verantwoordelijke(n) om de kwaliteit van de software te garanderen.

Inleiding

Een beeld zegt meer dan duizend woorden. Dat vooral in een tijdperk waar dataopslag exponentieel toeneemt in hoeveelheid en in belang. Bedrijven zijn steeds meer genoodzaakt om een manier te vinden om de verschillende soorten data te visualiseren om de waarde hieruit te halen. Hierdoor kunnen organisaties efficiënter tewerk gaan om de kwaliteit van hun producten en diensten te optimaliseren.

Data wordt binnen Selligent gecategoriseerd met als doel om externe doeleinden en voor interne doeleinden te bereiken. Binnen dit eindwerk wordt de focus gelegd op het visualiseren van interne data, meer specifiek de data die gerelateerd is aan softwareontwikkeling.

Bij softwareontwikkeling is er een proces dat uit verschillende stappen bestaat om tot een resultaat te komen. Het ontwikkelen van de software, testen van de verschillende functionaliteiten, *deployen* van de laatste versies, etc. Voor de organisatie is het belangrijk dat elke stap zo vlot mogelijk verloopt en dat de softwarekwaliteit zo hoog mogelijk is na het leveren van de laatste versies aan klanten.

Om de kwaliteit van het product te maximaliseren en de processen achter de productie te optimaliseren, wordt op zoek gegaan naar relevante en meetbare kwaliteitswaarden. Er wordt een lijst opgebouwd van verschillende kwaliteitswaarden met telkens een beschrijving over elke waarde. Daarnaast wordt gezocht naar de verschillende databronnen binnen Selligent waarin data bewaard wordt die gebruikt kan worden om de kwaliteitswaarden relevant te maken. Met behulp van deze data worden er *dashboarden* opgebouwd die trends in de tijd of cijfers op een huidig moment kunnen aantonen. Omdat softwarekwaliteit een verantwoordelijkheid is van elk individu die werkt in verband met softwareproductie, moet dit *dashboard* een algemeen beeld geven over het product. Daarnaast worden er ook *dashboarden* opgebouwd die voor andere functies binnen de organisatie relevant kunnen zijn. Zo wordt een *dashboard* gebouwd voor een algemeen team overzicht, een *dashboard* voor een overzicht van de *releases* en een eventueel advies over hoe een *dashboard* er uit kan zien indien *Customer Service Managers* betrokken moeten worden bij het proces.

De stage en het eindwerk trachten met behulp van duidelijke kwaliteitswaarden *dashboarden* op te stellen die als instrument dienen om de kwaliteit van zowel proces al product op te volgen en te verbeteren.

I. Stageverslag

1 Bedrijfsvoorstelling

De stage en het onderzoek worden uitgevoerd bij het internationale bedrijf Selligent, rond het product "Selligent Marketing Cloud". Dat is een marketing automatisatie tool die het leven van *marketeers* binnen grote bedrijven (kranten, media, *retail*-ketens, etc.) eenvoudiger maakt. De tool laat toe om op basis van profieldata en gebruikersgedrag een persoonlijke marketingboodschap te zenden via verschillende kanalen (e-mail, mobiele apps, etc.), en vervolgens de effectiviteit (aantal *clicks*, groei van het publiek, etc.) hiervan op te volgen. Het resultaat van de stage zal waar mogelijk rechtstreeks geïntegreerd worden in de processen en in de interne *cloud*-gebaseerde *dashboarding pipeline*.

Selligent Marketing Cloud is een bedrijf dat zich internationaal gevestigd heeft op locaties als de Verenigde Staten van Amerika, India, Duitsland, etc. In België is een kantoor gevestigd op de Corda Campus te Hasselt.

Figuur 1: Locatie kantoor van Selligent te Hasselt

2 Voorstelling Stageopdracht

2.1 Situering stageopdracht

Binnen Selligent is er een vraag naar een uniforme manier om de kwaliteit van hun product en hun productieproces in kaart te brengen. De visie binnen het bedrijf is om te streven naar software zonder bugs. Omdat er over vaste perioden bepaalde doelen ingesteld zijn om te behalen, wil men binnen het bedrijf weten wat allemaal relevant is om software kwaliteit te bepalen. Selligent wil op het einde van de stageopdracht weten:

In welke staat is de kwaliteit van de producten die opgebouwd worden door Selligent? En in welke plaatsen in het productieproces is er plaats voor verbetering om de proceskwaliteit te verbeteren?

2.1 Inleiding stageopdracht

Als uitvoering wordt gestart met het onderzoek naar wat softwarekwaliteit betekent en welke waarden er opgemeten kunnen worden om de softwarekwaliteit binnen Selligent in kaart te brengen. In het stageverslag wordt gewerkt met kennis die aangeworven is tijdens het onderzoek naar softwarekwaliteit. Er wordt regelmatig in het verslag gegrepen naar kwaliteitswaarden die verwoord zijn in het onderdeel '*II. Onderzoekstopic*'.

2.2 Uitwerking stageopdracht

De opdracht is onderzoek naar de waarden die relevant zijn om softwarekwaliteit te bepalen. Nadat er met behulp van verschillende bronnen een lijst opgesteld is waarin alle relevante kwaliteitswaarden bepaald zijn, wordt een selectie gemaakt van de meest relevante waarden.

Om te begrijpen wat softwarekwaliteit betekent voor Selligent, moet er naast bronnenonderzoek ook met stakeholders gepraat worden over wat voor hun van belang is om in kaart te brengen. Zo wordt er gepraat met *Software-engineers* van verschillende teams en worden er meetings gehouden met personen uit verschillende IT-domeinen (*testen*, *development*, ect.) om te begrijpen hoe de cohesie tussen de stakeholders zijn en wat de algemene belangen zijn binnen de organisatie.

Nadat gekend is wat softwarekwaliteit betekend voor de stagiair en de stakeholders die meehelpen bij dit project, moet er een visualisatie gebeuren met een tool naar keuze. Met behulp van de tool wordt een *dashboard* opgesteld. Dat *dashboard* bestaat uit een selectie van de meest relevante kwaliteitswaarden die in kaart moeten gebracht worden om op een efficiëntere manier de kwaliteit te verbeteren van de software bij Selligent.

Daarnaast moet er ook gekeken worden naar welke data er al reeds beschikbaar is om deze waarden te visualiseren. Indien er essentiële data niet beschikbaar is, moet er een aanbeveling gebeuren aan Selligent om additionele data te verzamelen of beschikbaar te stellen om overige waarden te visualiseren.

Naast het uitwerken van een kwaliteitsdashboard worden ook alternatieve opdrachten meegegeven om meerdere *dashboards* op te bouwen om andere functies te helpen. Zo wordt een *Releasedashboard, procesdashboard* en een *mockup* voor Sales opgebouwd die verder kunnen helpen bij het optimaliseren van het werk bij Selligent.

Het *releasedashboard* moet een algemeen overzicht geven van de software per release die er uitgevoerd wordt. Dat *dashboard* wordt uitgewerkt in Power BI. Dat omdat veel kwaliteitswaarde die gevonden zijn tijdens het onderzoek gebruikt kunnen worden binnen deze tool. Het *procesdashboard* dient om elk team binnen de organisatie te helpen bij het opvolgen van de vooruitgang tijdens sprints. Dat *dashboard* wordt opgebouwd in Azure DevOps om een vergelijking te kunnen maken tussen Power BI en Azure DevOps. Het *salesdashboard* wordt enkel uitgetekend om een beeld te krijgen van hoe dit *dashboard* er uit kan zien. De keuze om dit niet uit te werken komt doordat er te weinig kwaliteitswaarden gerelateerd of relevant zijn voor de salesafdeling.

Daarnaast wordt er een *dashboard* opgesteld die ook gebruik maakt van de lijst aan kwaliteitswaarden. Dit *dashboard* wordt gebruikt om verschillende partijen binnen Selligent te informeren over specifieke waarden. Het *dashboard* wordt uitgelegd door de verschillende onderdelen beknopt te beschrijven.

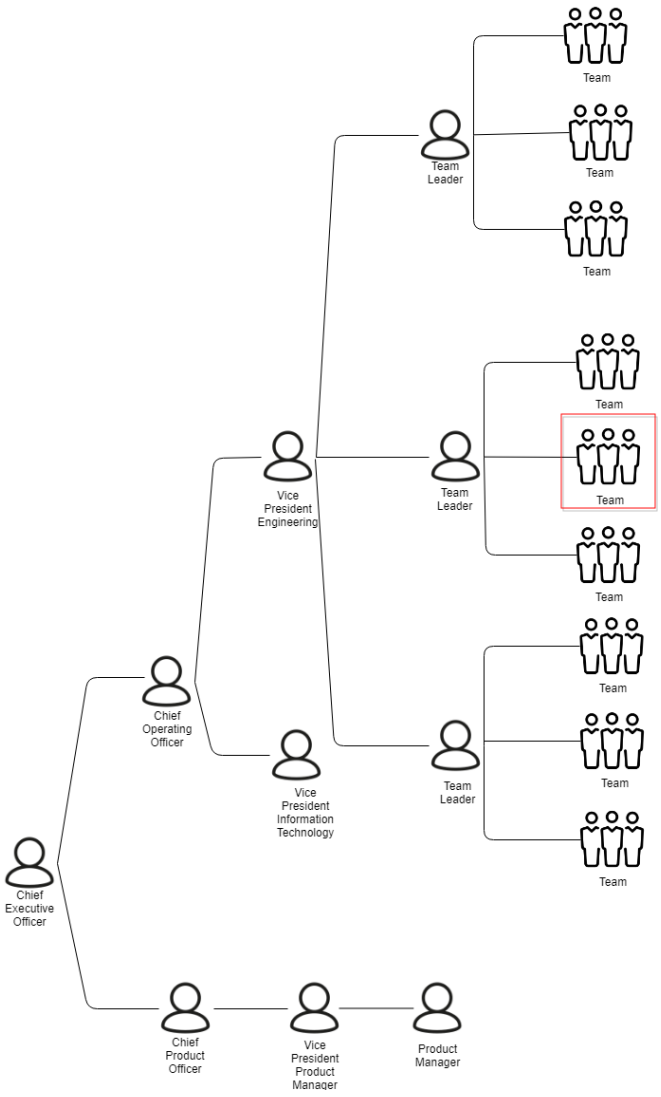
Als laatste opdracht wordt er een rapport opgesteld dat de kwaliteit van automatische testen moet aantonen. Op basis van dit rapport kan er geanalyseerd worden hoe de kwaliteit er voor staat op gebied van het verminderen van manuele testen en meer in te investeren in automatisering er van.

3 Bedrijfshiërarchie

De hiërarchie binnen Selligent loopt doorheen de verschillende kantoorlocaties. In Hasselt, waar de stage doorgaat, zijn er teams die bestaan uit *Software-engineers* die elk een eigen verantwoordelijkheid hebben binnen het team. Zo zijn er personen die bezig zijn met de applicatieontwikkeling, dataverwerking en zijn er personen aangesteld om de software te testen. Elk team staat onder leiding van een *Team Leader*. Die titel wordt ingevuld door één persoon maar heeft meerdere teams om het werk bij te houden. Op zijn beurt staat een *Director Engineer* onder leiding van een *Vice President.* De verdere hiërarchie kan teruggevonden worden in de afbeelding maar is van minder belang om te weten.

De focus van dit onderzoek is voor ieder persoon binnen de organisatie van belang en kan de betrokkenheid van meerdere teams, de *team leaders* en de *Vice President* opvragen indien deze nodig is om bepaalde antwoorden te verkrijgen.

De namen van de teams en van personen die functies invullen worden beschouwd als interne informatie. De functies binnen de organisatie worden in dit eindwerk niet verder besproken. Het rode vierkant rond een team representeert te situering van de stageopdracht.



Figuur 1:Organigram Selligent.

4 Procespipeline

Om te begrijpen waar softwarekwaliteit zich afspeelt, is het nodig om het productieproces te begrijpen binnen de organisatie. Zo kan er op meerdere plekken in het proces verdiepend worden gezocht naar meetbare waarden om te visualiseren.

Startende uit een *build*, wordt een nieuwe of geüpdatete feature samengebracht met de *masterbranch* tot een nieuwe *build*. Die *build* wordt bij gehouden met behulp van versiebeheer om bij fouten gemakkelijker terug te grijpen naar oudere versies om zo verder te kunnen werken op stabielere versies van de applicatie.

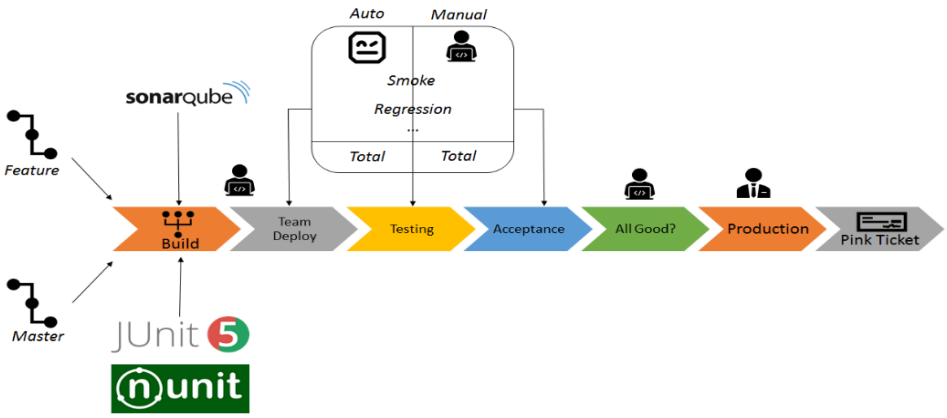
Wanneer een *build* uitgevoerd is, wordt met behulp van *unit* testen gecontroleerd of de functionaliteiten geen bugs of andere fouten bezitten. Tegelijk met de *unit* testen wordt met behulp van een code analysetool (bij Selligent is dit SonarQube) gekeken naar de kwaliteit van de code. [10]

Na de evaluatie met *unit* testen en een rapportuitslag van SonarQube wordt de *build* doorgeschoven naar een *team deployment* waarbij er uitgebreidere testen gedaan worden om de functionaliteiten te testen. Tijdens deze fase van het proces wordt er aan *smoke* testen en regressietesten gewerkt. De test *cases* tijdens deze fase bestaan uit zowel manuele als automatische test *cases*. De verhouding wordt bijgehouden tussen het aantal manuele en automatische testen. Dat wordt gedaan om de progressie naar automatisch testen bij te houden.

Tijdens *smoke* testen wordt nagegaan of de belangrijkste functies waar gebruikers mee gaan omgaan werken. Het resultaat van deze testfase geeft aan of een *build* stabiel genoeg is om door te voeren naar de volgende fasen. De resultaten van automatische testen worden bijgehouden in het Robot Framework. [9]

Regressietesten controleren of er een impact op reeds gebouwde functies is wanneer er nieuwe onderdelen aan toegevoegd worden. Zo wordt gecontroleerd op wijzigingen of niet werkende onderdelen van de applicatie. Die testen worden gedeeltelijk manueel als automatisch uitgevoerd. [6][17]

Nadat de regressie- en *smoke* testfase met succes afgerond zijn, wordt een finale *acceptance* test gedaan waarbij gekeken wordt naar het algemeen gebruik van de build en of deze voldoet aan de *requirements*. Als alles vanaf dan in orde is, wordt de *build* doorgevoerd naar productie waar eindgebruikers kunnen werken met de laatste *build*. Indien gebruikers toch nog problemen ondervinden, kunnen deze een *pink ticket* aanmaken dat in de *backlog* terechtkomt van *engineering*.



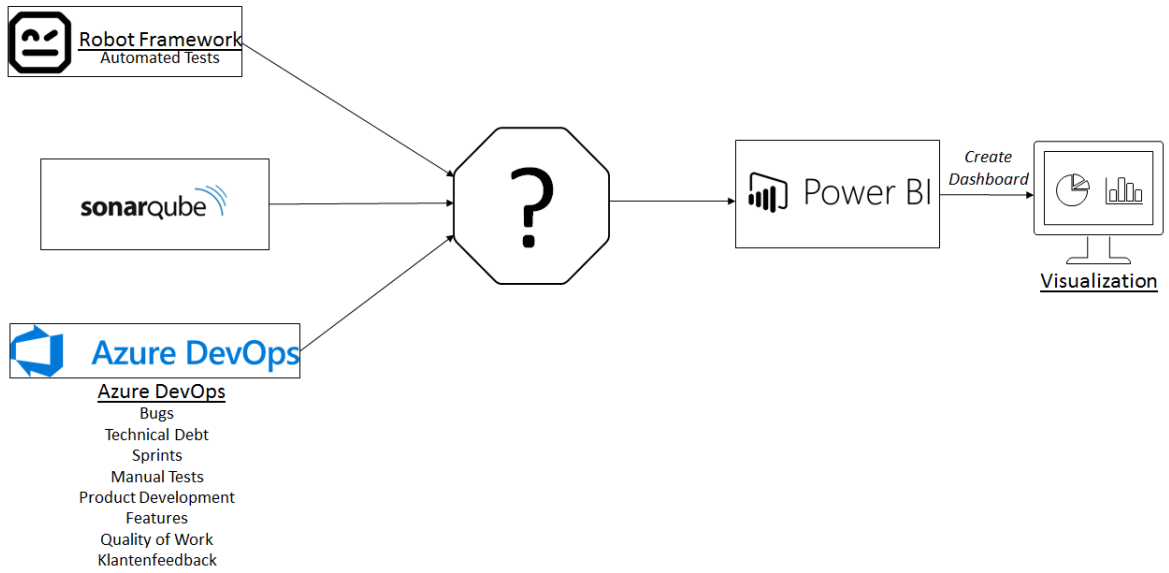
Figuur 2: Procespipeline aangevuld met testtools die gebruikt worden binnen Selligent.

5 Databronnen

5.1 Inleiding

We onderscheiden drie hoofdlocaties waar proces- en productinformatie opgeslagen wordt. Allereerst is er Azure Devops, waarin informatie bijgehouden wordt m.b.t. het agile ontwikkelingsproces over de tijd heen. Vervolgens is er Robot Framework, hetgeen automatische testen uitvoert om zo informatie rond productkwaliteit te bekomen. De resultaten van de automatische testen worden op een specifieke server opgeslagen. Tenslotte is er SonarQube dat een inzicht biedt in de kwaliteit van de code (bv. code *complexity* en code *smells*). Ook deze informatie wordt op een aparte server opgeslagen.

In dit hoofdstuk wordt inzicht gegeven in de data die beschikbaar gesteld wordt via verschillende tools en de manier waarop deze beschikbaar gemaakt wordt.



Figuur 3:Schematische voorstelling toolpipeline.

5.2 Azure DevOps

5.2.1 *Introductie*

Azure DevOps is een applicatie van Microsoft die gebruikt wordt om de werkstromen binnen applicatieontwikkeling binnen één platform te verwerken. Zo kunnen er *backlogs*, *repository's, pipelines, test cases* en vele andere essentiële onderdelen binnen applicatieontwikkeling verwerkt worden. Dat geeft als voordeel dat er heel veel data beschikbaar is op één platform.

5.2.2 *Databeschikbaarheid*

Van de kwaliteitswaarden die verwoord zijn in het onderzoekstopic, is er 47% van deze waarden beschikbaar in Azure DevOps. Wat voornamelijk beschikbaar is binnen Azure Devops, zijn de kwaliteitswaarden die gericht zijn naar bijhouden van gegevens over de *work items* die bijgehouden worden in de *backlog*. Er wordt op een uniforme manier bijgehouden wat de ernst, werkballast, staat en andere gegevens zijn voor verschillende types van *work items*. De itemonderverdeling bestaat uit: bugs, *features*, *technical debt*, *pink tickets*, enz. De onderverdeling bestaat uit een lijst van items waarvan niet alles even relevant is binnen de scope van het onderzoek.

Daarnaast is Azure DevOps geschikt om informatie te verkrijgen die gelinkt is aan sprints. Voor elke sprint wordt uitgebreid bijgehouden per team welke items er gelinkt zijn per vorige, huidige en toekomstige sprints. Daarnaast wordt elk *item* dat aanwezig is in de *backlog* gelinkt aan een specifieke sprint (dit kan afgeleid worden uit bijlage A).

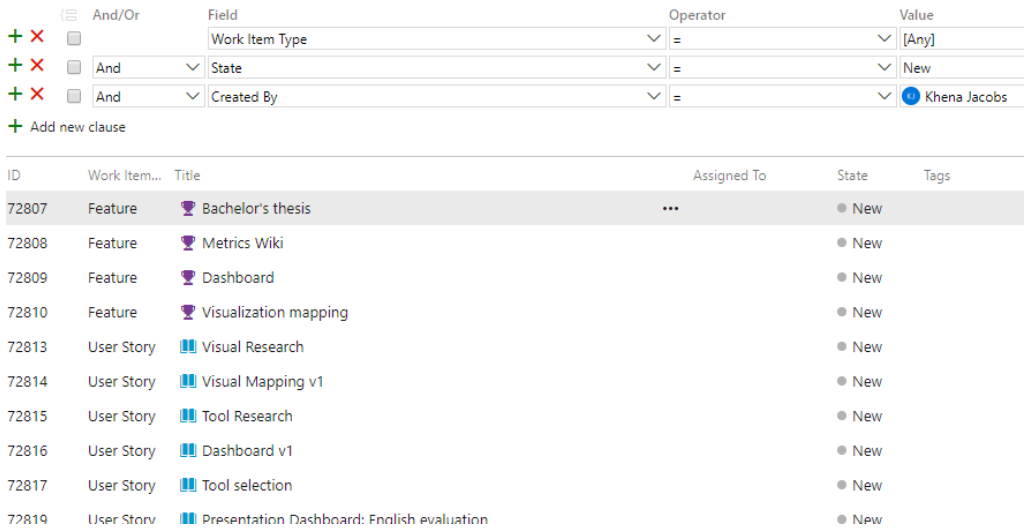
Wat niet beschikbaar is op het platform telt mee voor 38% van de kwaliteitswaarden. Ook zijn er bepaalde waarden wel af te lezen in Azure DevOps, maar kunnen deze niet uitgelezen worden met een tool. De kwaliteitswaarden die onder deze categorie vallen tellen mee voor 15%. Waarden van deze twee categorieën zijn de resultaten die gericht naar de kwaliteit van de code en de resultaten van de automatische testen. De code kwaliteit wordt niet gecontroleerd zoals het gedaan wordt in SonarQube. Er wordt enkel bijgehouden of een *build* slaagt of faalt. De testresultaten van automatische testen worden bijgehouden binnen de organisatie, maar de informatie die bijgehouden wordt is niet compleet. De incompleetheid wordt bepaald op basis van welke informatie nodig is om de resultaten te kunnen visualiseren. Die kwaliteitswaarden zullen aan de hand van andere databronnen moeten opgehaald en verwerkt worden.

5.2.3 *Data verzamelen uit Azure DevOps*

Om data te verkrijgen uit DevOps om deze te verwerken in Power BI, zijn er twee verschillende manieren. Een eerste manier bestaat uit het opstellen van query's in Azure DevOps, waarbij er een selectie gemaakt wordt aan *work items* op basis van de query voorwaarden. Nadat de query's opgesteld worden, kan er met behulp van een REST API de data opgevraagd worden en geformatteerd worden naar het gewenste formaat. Een tweede manier is door een *analytics view* op te stellen die net als query's werkt, maar deze opstelt zodanig dat Power BI deze automatisch kan structureren.

Query's en REST API

Door te werken met de REST API van Azure DevOps, moet er met behulp van query's in DevOps gewerkt worden. Via het venster dat aangetoond wordt, kan er een selectie opgesteld worden van de gewenste *work items* die bijgehouden moeten worden. De query gebeurt volgens de *Work Item Query Language* (*WIQL)* en wordt eenvoudig gebruikt door in de *drop down*-velden voorwaarden te selecteren en outputwaarden in te vullen. Als output wordt onderaan het scherm een lijst weergegeven van de *work items* die voldoen aan de voorwaarden. [16]



Figuur 4: Query scherm in Azure DevOps. (Dummy data)

Nadat de query aangemaakt is, kan de ingebouwde dashboardfunctie gebruikt worden in Azure DevOps. Die functie biedt de mogelijkheid om een overzicht van de werkomgeving in Azure DevOps te geven op basis van de opgestelde query's.

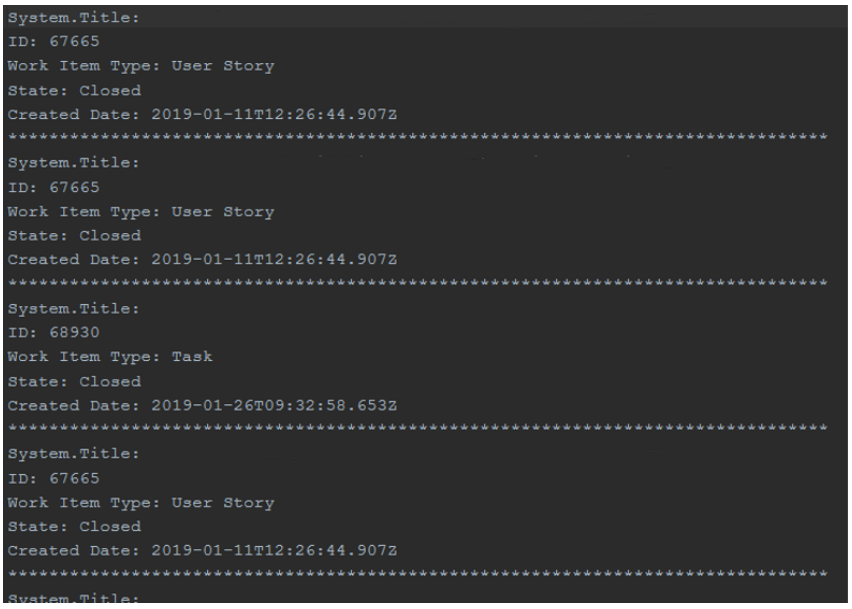
Voor een overzicht van de productkwaliteit kunnen *dashboards* die opgebouwd zijn in Azure DevOps voldoende zijn. Ook zijn er automatische en onmiddellijke wijzigingen aan resultaten en *dashboards* wanneer er binnen de werkomgeving aanpassingen gebeuren.

Om met de query's te werken binnen Power BI, moet er een omweg gemaakt worden die niet nodig is met *analytics view*. Om gegevens van de query op te vragen, moet er met de REST API gewerkt worden om de gegevens uit Azure DevOps op te vragen.

Azure DevOps beschikt over een REST API met de officiële documentatie van hoe de REST API gebruikt moet worden. Voor dit onderzoek is er een REST API geschreven in Python.

Wanneer er met een script informatie van een query opgevraagd wordt, kan er een lijst met *work items* opgesteld worden die gelinkt zijn aan de genoemde query. Enkel *work item ID* van *work items* kan getoond worden via deze methode. De informatie die opgevraagd wordt bij Azure DevOps wordt teruggegeven in een JSON-formaat. Om veldwaarden terug te krijgen, moet er uit het JSON-bestand geselecteerd worden welke relevant zijn.

Om de uitgebreide informatie te krijgen voor alle *work items* gelinkt aan een query. Moet er eerst een lijst opgesteld worden van de *work item ID's*. Daarna moet er via de API voor elke ID de informatie opgevraagd worden die nodig is. Als output wordt volgend .txt-bestand opgesteld waarbij er voor een query de naam, ID, *work item type,* status en aanmaakdatum van elk *work item* opgevraagd worden:



Figuur 5: Output query items met behulp van API.

Wanneer het bestand aangemaakt is, kan dit bestand ingeladen worden in Power BI. Power BI biedt de mogelijkheid om txt-bestanden in te laden. Power BI herkent de waardeverschillen die opgesplitst worden met ":". Echter zijn deze niet volledig accuraat. Om het bestand volledig bruikbaar te maken, moet er de nodige data *cleaning* gebeuren.

System.Title			_1
ID	67665	null	
Work Item Type	User Story	null	
State	Closed	null	
Created Date	2019-01-11T12	26	44.907Z
**********		null	
System.Title		null	
ID	67665	null	
Work Item Type	User Story	null	
State	Closed	null	
Created Date	2019-01-11T12	26	44.907Z
**********		null	
System.Title		null	
ID	68930	null	
Work Item Type	Task	null	
State	Closed	null	
Created Date	2019-01-26T09	32	58.653Z
**********		null	
System.Title		null	
ID	67665	null	
Work Item Type	User Story	null	

Figuur 6: Automatische output Power BI van .txt-bestand.

Data cleaning gebeurt in Power BI onder een tabblad waar alle *datasets* verzameld worden. Daarin is er een functie *Edit Query* die de mogelijkheid biedt om kolommen en rijen te bewerken naar wat de gebruiker wil als eindresultaat.

Via een REST API de data opvragen van DevOps heeft als voordeel dat de laadtijden en updates van de data vlotter verlopen dan als er met *analytics views* gewerkt wordt in Power BI. Indien de organisatie het essentieel vindt om zo snel mogelijk gegevens te kunnen verkrijgen van wat er gebeurt binnen de applicatieontwikkeling, is dit een methode die aangeraden wordt om te gebruiken.

Een nadeel dat bij deze methode vasthangt is de beperktheid van documentatie en informatie. Indien er onduidelijkheden zijn over de REST-API, zijn er een beperkt aantal bronnen die kunnen helpen bij het vinden van oplossingen. Het merendeel wordt teruggevonden in de officiële documentatie van Microsoft**.** [23] Ook is er een beperktheid in het opvragen van data uit DevOps. In vergelijking met *analytics view*, moeten voor de hand liggende berekeningen als *lead time* en *cycle* time nog manueel geïmplementeerd worden in de geschreven API. Daarnaast blijkt ook dat niet alle velden die terug te vinden zijn in Azure DevOps beschikbaar zijn via de geschreven API. Voor onder andere *bugs* is er geen manier om de *Closed Date* op te vragen. Die waarde is van belang om zowel *lead time* als *cycle* time te kunnen berekenen.

Concreet is het gebruik van query's en werken met een API geschikt als organisaties zoeken naar een oplossing om uit Azure DevOps live data op te vragen. Het verkrijgen van de data gebeurt sneller via een API dan via *analytics view*. Enkel is deze manier geschikt wanneer er data nodig is die wel toegankelijk is via de REST-API. Daarnaast moet er ook tijd vrijgemaakt worden om zelf een API uit te schrijven die het mogelijk maakt om alle gegevens per query op te vragen.

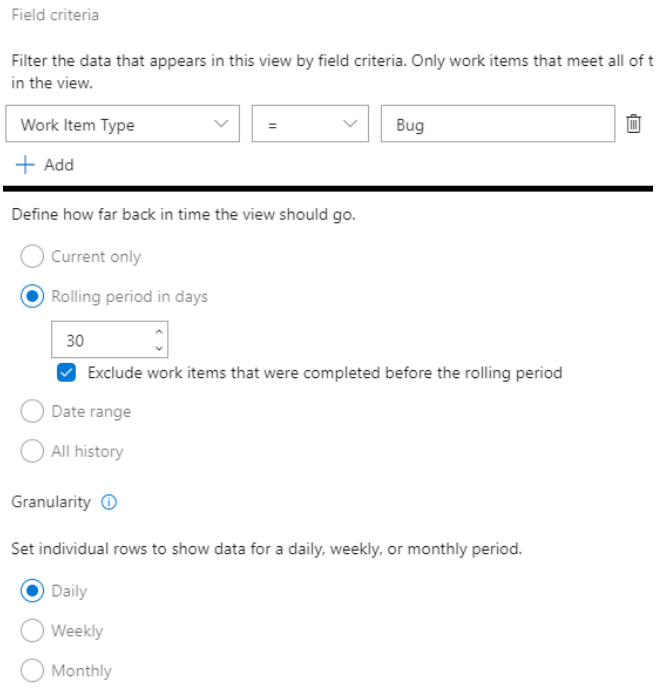
Analytics View

De tweede manier om data uit Azure DevOps te verkrijgen is via *analytics view*. Dat is een optie binnen Azure DevOps die soortgelijk is aan het opstellen van query's en kan bereikt worden via *Overview*. *Analytics* view is ook gebaseerd op WIQL en verwerkt de voorwaarden identiek als hoe het bij query's gebeurt. Het verschil is dat er geen automatische aanvulling is van waarden wanneer er naar specifieke outputwaarden gezocht worden.

Naast het zoeken naar veldwaarden kan er ook meegegeven worden over welke tijdspanne er data gezocht moet worden. Ook is het mogelijk om per team de *work items* op te vragen indien dit nodig is.

Als voorbeeld:

Er worden *work items* gezocht van het type Bug en zijn aangemaakt in de laatste 30 dagen. In *Fields* wordt het *Item Type* ingevuld als Bug en wordt in *History* gevraagd naar de laatste 30 dagen.



Figuur 7: Invullen van voorwaarden in Analytics View.

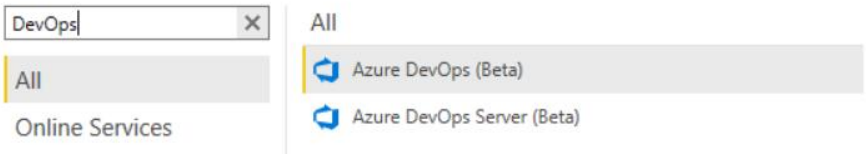
Na het verwerken van de voorwaarden moet er een controle uitgevoerd worden van de *view*. Hierbij controleert Azure DevOps of de *view* correct aangemaakt is en bruikbaar is voor Power BI. De verificatie voor Power BI is nodig om de automatische structurering van de tabel met data mogelijk te maken. Afhankelijk van hoeveel data en van de tijdsperiode waarvan data opgevraagd wordt, kan de verificatietijd verschillen tussen enkele seconden tot vijf minuten. Na de verificatie wordt het aantal rijen getoond waarmee gewerkt kan worden in Power BI.

Your analytics view has been verified and is ready to use with Power BI.
Learn more about the Azure DevOps data connector.

Estimated rows: 23820

Figuur 8: Verificatie voor Power BI.

Nadat de *analytics view* geverifieerd is om te gebruiken in Power BI, kan er via Power BI een connectie gemaakt worden met de Azure DevOps werkomgeving. Power BI biedt de mogelijkheid om rechtstreeks uit Azure DevOps data op te vragen via de *analytics views*.



Figuur 9: Selectie van Azure DevOps als databron.

Nadat de link gelegd is tussen het Azure DevOps account en Power BI, kan de aangemaakte *analytics* view ingeladen worden in Power BI en is het opstellen van de structuur al reeds gebeurd. Enkel de datatypes moeten in de kolomheaders gewijzigd worden naar het juiste type. Alle datawijzigingen moeten gebeuren aan de hand van *data cleaning.*

5.3 Data verzamelen uit Robot Framework

Robot Framework is een *open-source* automatisering *framework* dat gefocust is op *smoke testen*. Omdat data gerelateerd met de automatische testen eerder beperkt is te vinden in Azure DevOps, moet er rechtstreeks uit het Robot Framework data opgehaald worden. De resultaten van testen worden bijgehouden per *build* die getest is. Elke *build* krijgt in de titel ook de datum mee. In de folder kan er een HTML-bestand en XML-bestand teruggevonden worden waarin de resultaten van de testen zich bevinden. Om de resultaten in een leesbaar formaat te krijgen om er met Power BI aan de slag te kunnen gaan, moet er een script geschreven worden om via een API output uit de testen te halen. De output van de testen worden bijgehouden in het XML-bestand. Per test case ziet het er als volgt uit:

"<kw name="Connect To Database Using Custom Params" library="DatabaseLibrary"><doc>Loads the DB API 2.0 module given `dbapiModuleName` then uses it to</doc><arguments><arg>pymssql</arg><arg>${dbConnString_Master}</arg></arguments><msg timestamp="20190207 07:11:12.228" level="INFO">Executing : Connect To Database Using Custom Params : pymssql.connect(db_api_2.connect(host="AZXSQL01N01.emsecure.local", database="EGS_TEST_MASTER", user="DBA_TEST",password="BAxpDCbG1rGJI2Aa1LgJ")) </msg><status status="PASS" endtime="20190207 07:11:12.384" starttime="20190207 07:11:12.196"></status>
</kw>"

Via het script wordt de informatie uit de *tags* gehaald. Het script wordt geschreven in Python en wordt uit alle *build directories* de resultaten per naam van de *build* gesorteerd in een CSV-bestand. Voor elke *build* wordt verwerkt wat het aantal geslaagde en gefaalde testen zijn voor API- en GUI-testen. Daarnaast is er ook het totaal geslaagde en gefaalde testen uit de XML bestanden gehaald. *File Name* is in het voorbeeld verwijdert om interne data te beschermen. Na het verwerken van de resultaten in het CSV-bestand, ziet de output er als volgt uit:

File Name	Total Tests	Total Passed	Total Failed	Percentage F	Total API Tes	API Passed	API Failed	Perc
	209	198	11	95	74	66	8	
	211	201	10	95	74	70	4	
	211	189	22	90	74	66	8	

Figuur 10: Output van automatische test cases van Robot Framework.

5.4 Toolpipeline

Dit onderdeel dient om alle beschreven databronnen in kaart te brengen en om aan te tonen hoe deze helpen tot het bekomen van een kwaliteitsdashboard in Power BI.

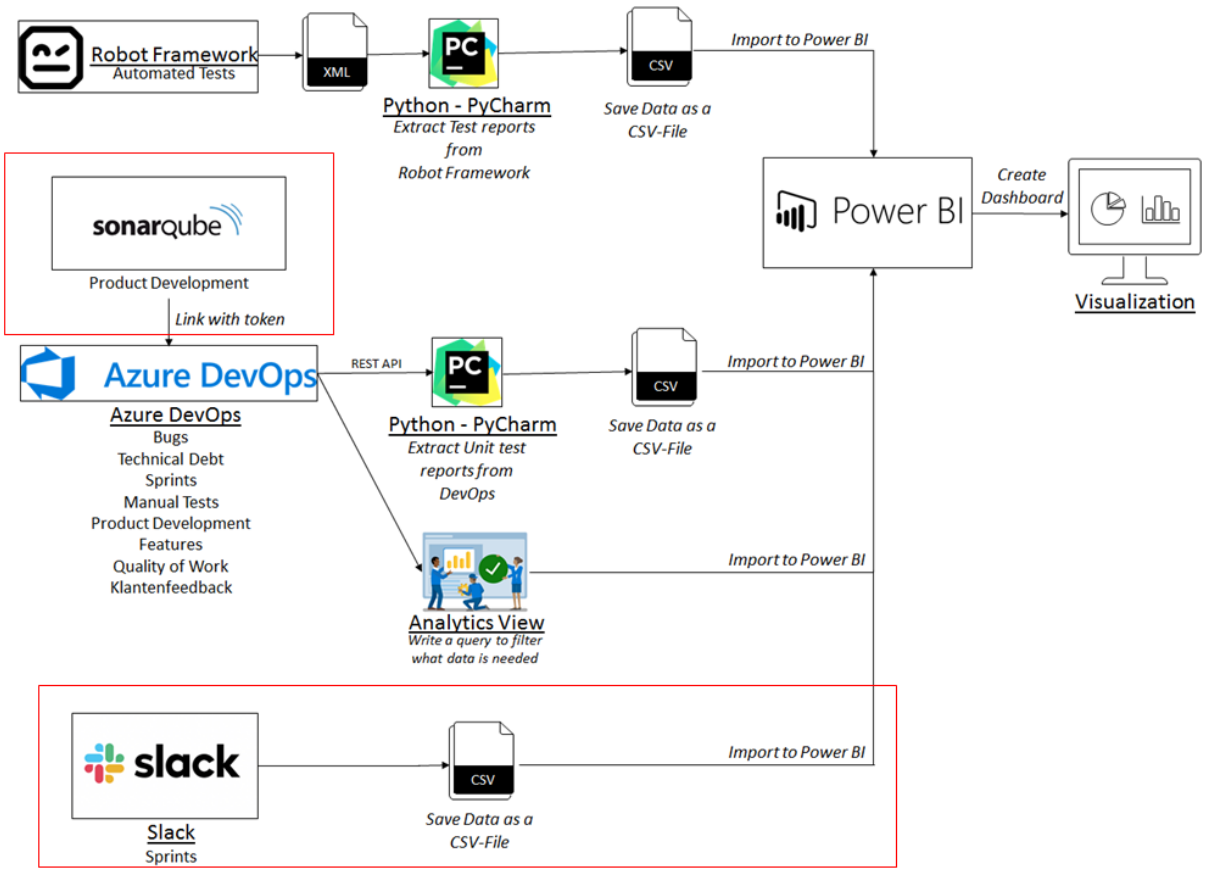
De resultaten die komen uit de automatische testen worden op een server bijgehouden met een onderverdeling in *stable*, *flaky* en algemene resultaten. Omdat het *dashboard* een algemeen beeld moet geven van de softwarekwaliteit, wordt uit de algemene resultaten data gehaald. Vanuit elke *build* wordt in het XML-bestand uitgelezen hoeveel testen er in totaal gebeuren, hoeveel testen slaagden en hoeveel faalden. Op basis van deze gegevens worden gemiddelden berekent. Dat alles gebeurt met behulp van een zelfgeschreven script in Python met behulp van PyCharm. Nadat de gegevens uit het Robot Framework verwerkt zijn, worden deze in een CSV-bestand bijgehouden. Hierna kan er in Power BI gewerkt worden met dit bestand.

Azure DevOps biedt de mogelijkheid om met *analytics view* een uitgebreide hoeveelheid data op te vragen die relevant is om te gebruiken in PowerBI. Data die niet toegankelijk is via *analytics view* is voornamelijk diepgaande informatie rond de *builds* en andere code-gerelateerde informatie. Om deze op te vragen moet er gewerkt worden met de REST API die Azure DevOps aanbiedt. Net als bij het Robot Framework wordt een zelfgeschreven script in Python gebruikt om informatie rond elke *build* op te vragen. Als beschreven in het onderzoeksonderwerp, is het belangrijk om de testfasen in kaart te brengen. Voor de *builds* wordt via het script het aantal *unit* testen opgevraagd en bewaard in een CSV-bestand. Daarna wordt dit bestand ingeladen in Power BI.

Omdat binnen Selligent SonarQube niet uitgebreid genoeg gebruikt wordt, is het niet mogelijk om de kwaliteitswaarden rond codekwaliteit te visualiseren. Voor het onderzoek is toch naar een mogelijkheid gezocht om SonarQube te linken aan Power BI. Zo is er een mogelijkheid om met behulp van een persoonlijke *token* een link te leggen tussen de rapporten in SonarQube en de resultaten in Azure DevOps.

Slack kan als alternatief gebruikt worden om de kwaliteitswaarde die staat voor de releasetevredenheid binnen Selligent in kaart te brengen. Deze tool is tijdens de stage uit de *scope* geplaatst omdat klantentevredenheid complexer is dan vooraf ingeschat.

Op de volgende bladzijde is er een schematische voorstelling te zien van de verwerking van alle databronnen om deze in één geheel te gebruiken.



Figuur 11:Schematische voorstelling toolgebruik en ophalen van data.

6 Visualisatietools

6.1 Inleiding

Het doel van dit eindwerk is om inzicht te verkrijgen in de besproken kwaliteitswaarden (zie II.H3 *Kwaliteitswaarden*). Hiervoor zijn visualisaties nodig (II.H4 *Visualisatiemogelijkheden*) die gebouwd zijn op de aanwezige databronnen (I. H5 *Databronnen*). Om dit te vergemakkelijken is er een groot aanbod van applicaties beschikbaar die het eenvoudig maken om datasets uit verschillende bronnen voor te bereiden en weer te geven.

6.2 Toolkeuze

Aangezien er een enorm groot aanbod aan mogelijke applicaties zijn die datarapporten en *dasboards* kunnen opstellen, beperken we ons tot de opties die voor Selligent relevant zijn. Als eerste is er Azure Devops, dat reeds in gebruik is binnen Selligent en kan basisdashboards voorzien. Als tweede is er Power BI, hetgeen meer flexibele dashboards toelaat, maar wel nog steeds eenvoudig te integreren valt met DevOps en andere systemen. Als laatste beschouwen we Jupyter Notebooks, een eerder technische applicatie die door een *data scientist* gebruikt wordt om data zeer nauwkeurig te analyseren. De drie geselecteerde tools worden op zeven domeinen vergeleken op hun functionaliteiten en gebruiksvriendelijkheid.

6.3 Jupyter Notebook

6.3.1 *Introductie*

Plot.ly is een applicatie die gericht is naar het analyseren van data. In vergelijking met Power BI is het doel van Plot.ly om data uit alle domeinen te kunnen visualiseren. Plot.ly is een online webapplicatie die gebruik maakt van een Python *framework.* Daarbij zijn de *dashboards* op een interactieve manier te gebruiken. [19]

Pandas is een *library* in de Python programmeertaal. Pandas biedt die mogelijkheid aan om zowel data-analyses te doen als het bewerken van datasets. De keuze om deze in combinatie met Plot.ly te gebruiken is omdat deze samen een beter geheel vormen tegenover Power BI. Ook zijn niet alle datasets vanaf het begin dat ze opgehaald worden uit bronnen even bruikbaar. Die datasets moeten kunnen bewerkt worden zodat alle waarden bruikbaar en leesbaar zijn.

Om op een vlotte manier te kunnen werken met Pandas en Plot.ly, wordt er met Jupyter Notebook lokaal gewerkt.

6.3.2 *Instapniveau*

Om te begrijpen hoe er gewerkt moet worden met Pandas en Plot.ly, is het nodig om een begrip te krijgen in het programmeren in Python. Daarnaast is het ook nodig om te begrijpen hoe er gewerkt moet worden met de *library* en *tool* vooraleer er met data gewerkt kan worden. Denk aan de mogelijke functionaliteiten, de variabelen die meegegeven moeten worden met functies, etc. Omdat Pandas en Plot.ly veelgebruikt worden in de Python*community* en door een *data scientist,* is er veel hulp terug te vinden op blogs en forums die voldoende uitleg geven op vaak voorkomende vragen die naar boven komen bij nieuwe gebruikers.

6.3.3 *Voorbereiding*

Alvorens we met Plot.ly willen werken, is het noodzakelijk om een Jupyter Notebook werkomgeving te hebben. Dat help bij het ontvangen van resultaten nadat er code geschreven wordt.

Nadat er een werkomgeving beschikbaar is in Jupyter Notebook, moet er een account aangemaakt worden voor Plot.ly om de output van grafieken en tabellen te kunnen weergeven en opslaan.

6.3.4 *Data Cleaning*

Om data te verwerken wordt er gewerkt met Pandas. In vergelijking met de verwerkingsmogelijkheden van Plot.ly is dit efficiënter en gebruiksvriendelijker wanneer er een overzicht gemaakt wordt van de data. Plot.ly wordt in dit onderzoek gebruikt om de visualisaties te doen op basis van de data die bewerkt is met behulp van Pandas.

Ter vergelijking: Wanneer er eenzelfde dataset ingeladen wordt in Plot.ly en Pandas, wordt bij Pandas automatisch rekening gehouden met de lengte van de waarden in de cellen. Bij Plot.ly is er een vaste breedte in de kolommen die naderhand nog gewijzigd moet worden om dezelfde leesbaarheid te krijgen als bij Pandas. In de voorbeelden zijn gegevens verwijdert uit de screenshots om interne data te beschermen.

	EXTID	CREATED_DT	CLOSE_DT	PRIORITY	STATE	ANSWER_DURATION	ANSWERED	CLOSE_DURATION	NAME	DIVISIONID
0		2019-04-04T12:40:14.007	NaN	HIGH	NEW		0			NaN
1		2019-04-04T10:54:03.51	NaN	NORMAL	OPEN		0			29.0
2		2019-04-04T08:47:57.453	NaN	HIGH	NEW		0			NaN
3		2019-04-03T17:38:20.27	NaN	NORMAL	OPEN		0			29.0
4		2019-04-03T16:13:39.433	NaN	NORMAL	OPEN		0			29.0
5		2019-04-03T10:18:21.827	NaN	NORMAL	OPEN		0			29.0

File_Name	Total_Tests	Total_Passed	Total_Failed	Percentage_Total	API_Tests	API_Passed	API_Failed	Percentage_API	GUI_Tests	GUI_Passed	GUI_Failed	Percentage_GUI
nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
						[illegible]xml	8.0	89.0	135.0	132.0	3.0	98.0
nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
						[illegible]xml	4.0	95.0	137.0	131.0	6.0	96.0
nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
						[illegible]xml	8.0	89.0	137.0	123.0	14.0	90.0
nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
						[illegible]xml	0.0	100.0	27.0	21.0	6.0	78.0
nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan

Figuur 26: Bovenaan: Dataset met Plot.ly; Onderaan: Dataset met Pandas.

Na een rechtstreeks ontvangst van de data, valt er af te leiden dat sommige waarden geen leesbare formaal hebben in de cel. Data *cleaning* met Pandas heeft als voordeel dat er met één lijn code een hele kolom aan waarden gewijzigd kan worden. De wijzigingen gebeuren op de manier weergegeven in volgend voorbeeld:

```
df.NAME = df.NAME.replace(to_replace="-",value="No Name")
```

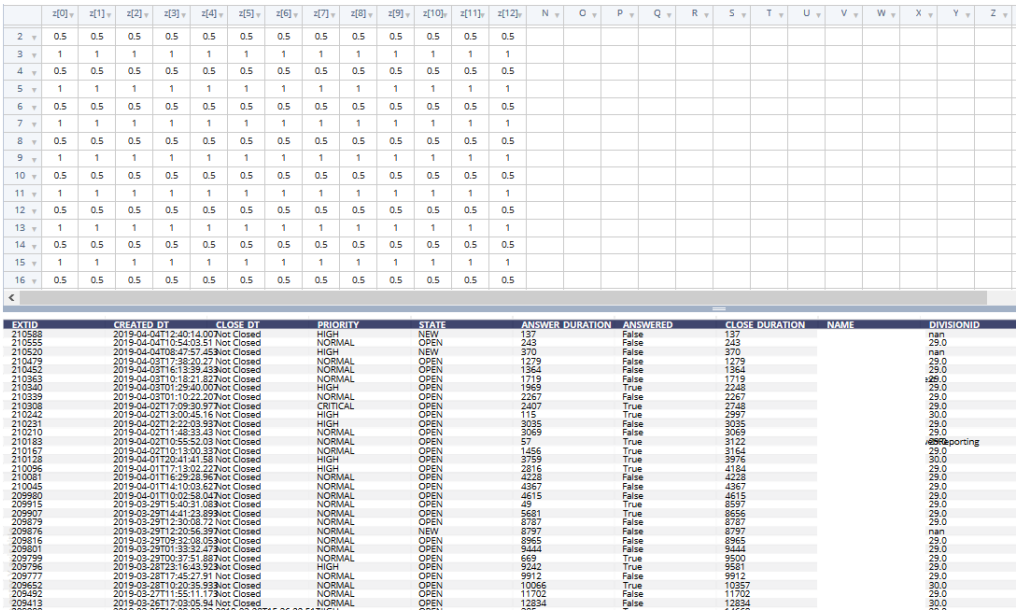
Het effect van de data *cleaning* is dat het bestand leesbaarder is voor de gebruiker. Ook is dit een hulp bij het visualiseren ervan. Interne data is uit het voorbeeld verwijderd om deze te beschermen.

	EXTID	CREATED_DT	CLOSE_DT	PRIORITY	STATE	ANSWER_DURATION	ANSWERED	CLOSE_DURATION	NAME	[
0	210588	2019-04-04T12:40:14.007	Not Closed	HIGH	NEW		False			
1	210555	2019-04-04T10:54:03.51	Not Closed	NORMAL	OPEN		False			
2	210520	2019-04-04T08:47:57.453	Not Closed	HIGH	NEW		False			
3	210479	2019-04-03T17:38:20.27	Not Closed	NORMAL	OPEN		False			
4	210452	2019-04-03T16:13:39.433	Not Closed	NORMAL	OPEN		False			

Figuur 27: Voorbeeld van resultaat data cleaning.

Nadat het bestand volledig bewerkt is om het bruikbaar te maken, kan de visualisatie mogelijk gemaakt worden.

Een tweede probleem dat opduikt als tabellen ingeladen worden in Plot.ly, is dat het bestand dat ingeladen wordt, beschouwd wordt als een grafiek of diagram. Dat is te zien wanneer er via de webapplicatie gezocht wordt naar de tabel. In onderstaande figuur is te zien dat op de webapplicatie van Plot.ly onderaan een toegevoegde tabel te zien is die geen betekenis heeft. Dat wordt gedaan terwijl het de bedoeld is om bovenaan de webapplicatie de tabel te krijgen. Om dit te vermijden moet het bestand manueel via de Plot.ly webapplicatie ingeladen worden.



Figuur 28: Webapplicatie Plot.ly: De foute data (bovenaan) die ingeladen wordt met de gewenste data (onderaan).

6.3.5 *Update Datasets*

Bij Pandas en Plot.ly moeten de *datasets* manueel binnengehaald worden in de omgeving. In tegenstelling tot de andere *tools* is er geen rechtstreeks implementatie van de databronnen. Dat zorgt er voor dat er telkens een met een eigen geschreven script data uit de bronnen gehaald moet worden en verwerkt worden in een leesbaar formaat voor Python en Plot.ly. voor het gebruikte voorbeeld van het Robot Framework, duurt het tot maximum 10 minuten om de *dataset* te updaten. Daarna moet de *dataset* ingeladen worden in Jupyter Notebook om daarna in het script van Pandas en Plot.ly de update door te voeren.

6.3.6 *Live Data*

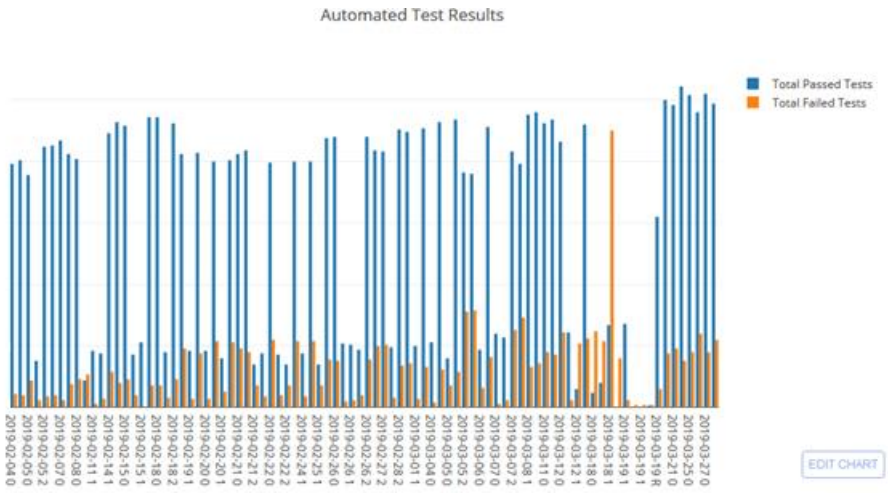
Voor Jupyter Notebook is er geen mogelijkheid om *live* data te visualiseren. In de eerste plaatst omdat er telkens manueel data geüpdatet moet worden uitgelegd in *Update Datasets.* Daarnaast bieden Plot.ly en Pandas geen mogelijkheid aan om data *live* te updaten.

6.3.7 *Visualiseren van waarden*

In volgend voorbeeld wordt kwaliteitswaarde 4.3.6 (*Automatische en Manuele testen)* van het onderzoekstopic in kaart gebracht. Hierbij wordt met behulp van een staafdiagram inzicht gegeven in de verhouding geslaagde en gefaalde automatische testen per *build*. Het bestand dat uitgelezen wordt is afkomstig uit een Robot Framework die de automatische testen bijhoudt.

In het XML-bestand van het Robot Framework staan gegevens die voor de specifieke kwaliteitswaarde minder relevant zijn. Er wordt aan data *cleaning* gedaan zodat er enkel de relevante informatie overblijft. Dat zorgt voor betere laadtijden bij het opvragen van de grafieken in Plot.ly.

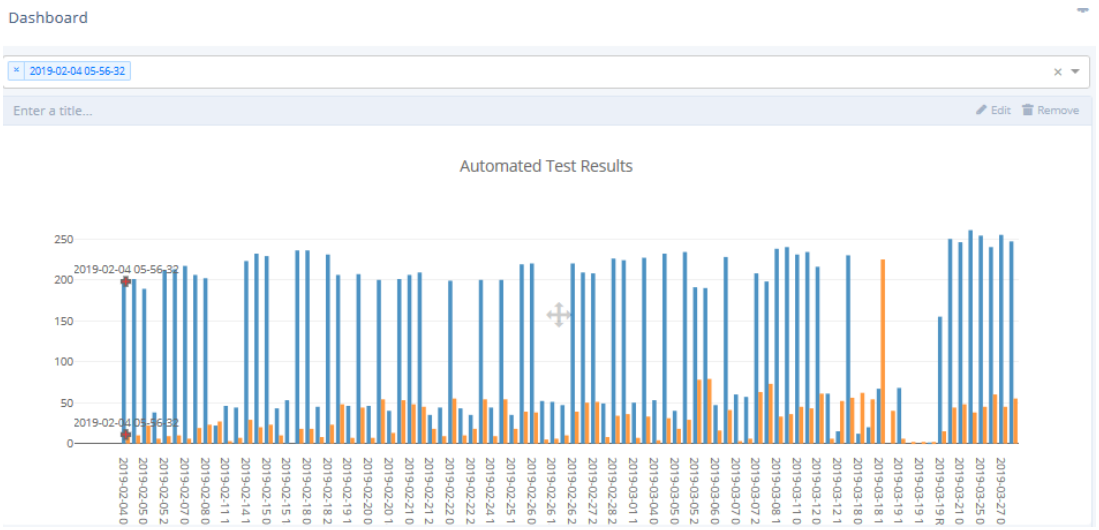
Om volgens de historische data in te kijken hoe de resultaten zijn van de automatische testen, moet er uit de titel van de *build* de datum gehaald worden. Dat moet gebeuren om een chronologische volgorde te kunnen plaatsen op de x-as van het staafdiagram. Nadat met Pandas de bewerkingen uitgevoerd zijn, wordt met Plot.ly een staafdiagram opgebouwd die volgende weergave toont. Waarden op de y-as zijn verwijdert om interne data te beschermen:



Figuur 29: Staafdiagram automatische testen.

Nadat de grafiek in Jupyter aangemaakt is, wordt deze automatisch opgeslagen op het Plot.ly account.

Plot.ly heeft als optie om extra zicht te krijgen op grafieken en diagrammen de mogelijkheid om snel en eenvoudig in te zoomen op bepaalde stukken van de grafiek of diagram. Dat gebeurt zonder laadtijden en heeft als effect dat er snel kan gewerkt worden met de verkregen output. Daarnaast is er op de webapplicatie de mogelijkheid om naar specifieke waarden te zoeken in een zoekbalk. Na het ingeven worden deze gemarkeerd op het diagram.



Figuur 30: Zoekfunctie Plot.ly en arcering gezochte waarde.

Een dashboard in Plot.ly is opgesteld gelijkaardig aan de methode die toegepast wordt in Tableau. Elk dashboard bestaat uit één grafiek of diagram die individueel opgesteld moet worden. Dat is verschillend in vergelijking met Power BI. Het voordeel van de manier waarop Plot.ly gebaseerd is, is dat er geen misleidende interpretaties kunnen gebeuren als het dashboard onduidelijk opgesteld is. Een nadeel is dat er een beperkt inzicht kan zijn.

Voor de productkwaliteit is het belangrijk om meerdere elementen naast elkaar te kunnen leggen. Veel opgenoemde waarden zijn relevant als ze vergeleken worden met andere waarden in een oogopslag.

6.3.8 *Gebruik van Plot.ly en Pandas*

Eens de manier van werken bij de gebruiker gekend is, kan de combinatie van Plot.ly en Pandas zeer krachtig zijn om aan data-analyse te doen. Wel moet er bewust een afscheiding in werk gedaan worden tussen Pandas en Plot.ly. Ter verduidelijking, zowel Plot.ly als Pandas beschikken over elkaars functionaliteiten. Enkel zijn deze functionaliteiten moeilijker in gebruik. Zoals beschreven bij data *cleaning*, zijn er bepaalde automatische zaken die gebeuren bij Pandas terwijl dit manueel moet gebeuren in Plot.ly.

Wanneer er na data *cleaning* output opgevraagd wordt van dit bestand, moet er rekening gehouden worden met Pandas die niet zoekt naar bewerkte waarden. Wanneer er uit een kolom waarden opgevraagd worden, moet er gezocht worden naar de oorspronkelijke data uit deze kolom. Dat kan voor een probleem zorgen indien er geen documentatie aanwezig is over de oorspronkelijke data. In onderstaand voorbeeld wordt er gezocht naar de klanten *tickets* die nog geen antwoord ontvangen hebben van de organisatie. De originele waarde op het ticket is “0” indien er geen antwoord is. De bewerkte waarde toont *False*. Wanneer er dus gezocht moet worden naar deze waarde, moet er naar “0” gezocht worden.

ANSWERED	ANSWERED
0	False
0	False
0	False
0	False
0	False

Figuur 32: Originele (links) en bewerkte (rechts) waarden kolom "Answered".

```
1 dfTest = df['ANSWERED']==0
2 dfTest
```

```
0     False
1     False
2     False
3     False
4     False
5     False
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13    False
14    False
15    False
```

```
1 dfTest = df['ANSWERED']=="False"
2 dfTest
```

```
0      True
1      True
2      True
3      True
4      True
5      True
6     False
7      True
8     False
9     False
10     True
11     True
12    False
13    False
14    False
15    False
```

Figuur 33: Zoekresultaat met originele(links) en bewerkte(rechts) waarden.

Plot.ly verwerkt datatypes die numerieke of tijdswaarden bevatten. Wanneer er met andere datatypes gewerkt wordt, moeten deze omgevormd worden tot een leesbaar type. Als dit niet gedaan wordt kan Plot.ly geen zichtbare grafiek of diagram weergeven en geeft het een leeg veld terug.

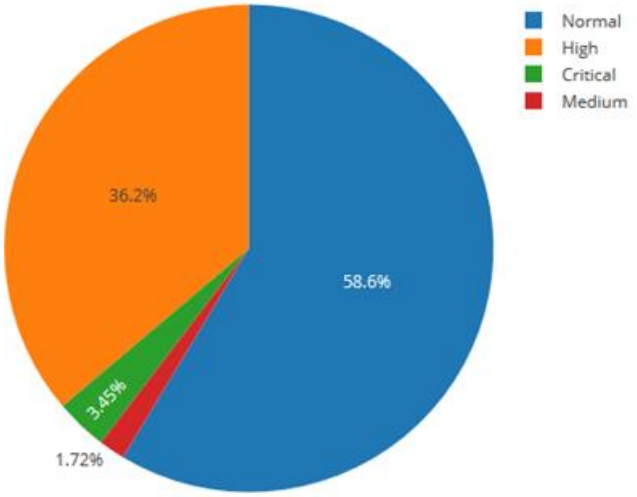
Als voorbeeld wordt uitgelegd welke conflicten er zijn alvorens er een visualisatie kan gebeuren van kwaliteitswaarde 4.8.6 (*Ernst van Customer Items*).

Wanneer het aantal *pink tickets* uitgelezen moeten worden, moet er data opgevraagd worden uit een JSON-bestand. Dat bestand moet met behulp van een Python script opgehaald worden en omgevormd worden tot een CSV-bestand. Nadat het CSV-bestand beschikbaar is, kan er met Pandas een tabel aangemaakt worden op basis van dit bestand. De data is dan zichtbaar als in Figuur 34. Om de waarden binnen elke kolom in een leesbaar formaat te krijgen, moet er met behulp van data *cleaning* met Pandas gewerkt worden.

Een eerste conflict dat opduikt wanneer het cirkeldiagram aangemaakt wordt, is dat Plot.ly geen Pandas datatypes kan inlezen. Bewerkte waarden worden als een “pandas.core.series.Series” type opgeslagen. In het voorbeeld wordt een beeld gecreëerd van alle types van ernst voor de *pink tickets* en het aantal per type van ernst. Om in Plot.ly dit leesbaar te maken, moet elke hoeveelheid per categorie omgevormd worden tot een integer.

Een tweede conflict is dat Pandas geen integer waarden kan bijhouden in een array. Dat zorgt dat er voor elke categorie van ernst (*Normal*, *Medium*, *High* en *Critical*) individueel geteld moet worden hoeveel keer deze voorkomt in de kolom ‘Priority’.

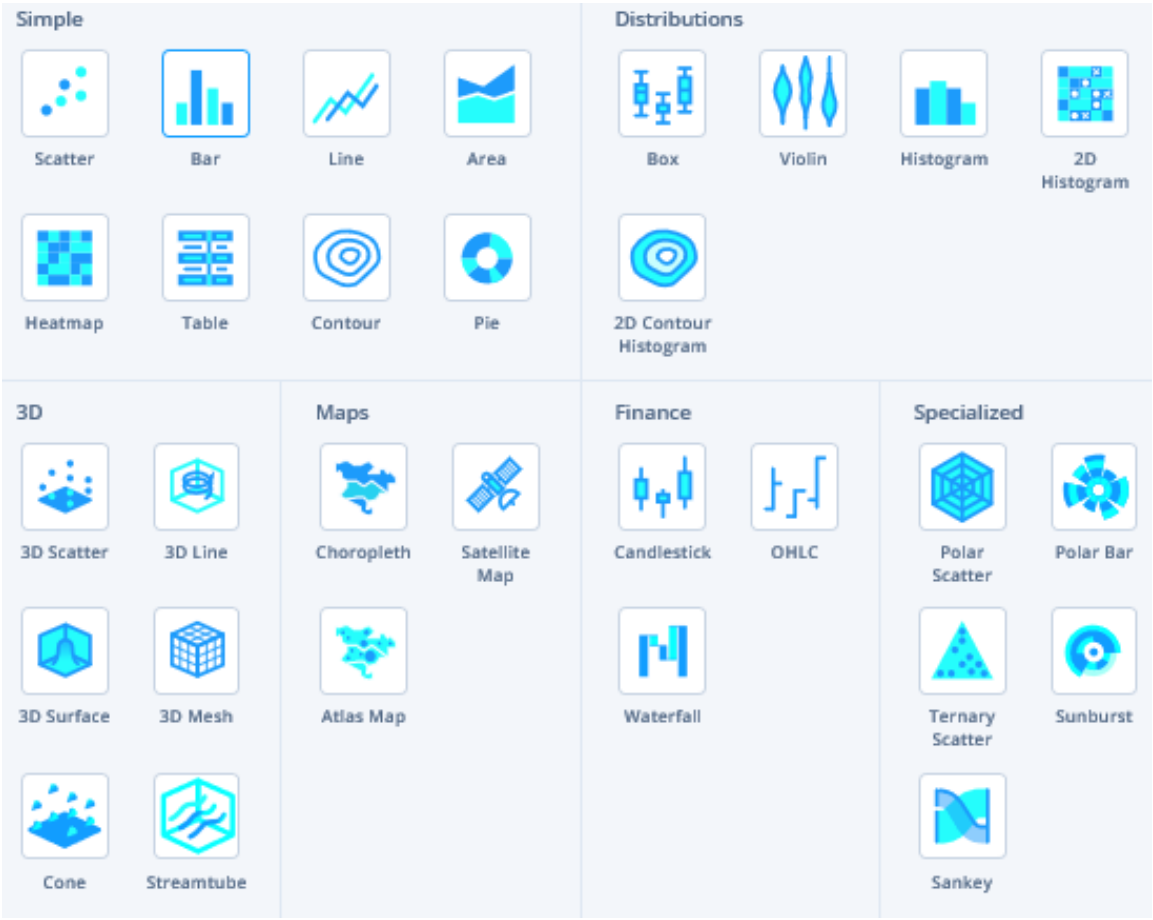
Voor Plotl.ly ligt het probleem bij de gebruiksvriendelijkheid en performantie van hun webapplicatie. Wanneer er grotere CSV- of XSLX-bestanden ingeladen worden, dan heeft dit een enorm effect op de laadtijd van wijzigingen die doorgevoerd moeten worden. Daarnaast is het uploaden van deze bestanden anders wanneer er via Jupyter en geschreven code een tabel ingeladen wordt dan dat dit gebeurt via de webapplicatie.



Figuur 34: Ernst van Customer Items gevisualiseerd met Plot.ly.

6.3.9 *Visualisatiemogelijkheden*

Bij het gebruik van Plot.ly is er een uitgebreide set aan visualisatiemogelijkheden. Die zijn gestructureerd te vinden op het online platform van Plot.ly. De lijst bevat alle mogelijke grafieken en diagrammen die het meeste gebruikt worden. Indien er een visualisatiemethode ontbreekt kan er niet naar een *marketplace* gegaan worden zoals bij de andere tools om uit een groter aanbod te selecteren.



Figuur 12: Visualisatiemogelijkheden Plot.ly

6.3.10 *Opbouw Dashboard*

Pandas en Plot.ly zijn visualisatietools die niet gericht zijn naar het opbouwen van een dashboard om meerdere grafieken en/of diagrammen te tonen. Om een overzicht te krijgen van meerdere waarden kan er in Jupyter Notebook een samenvatting gemaakt worden van alle grafieken. Dat heeft meer als functie om een rapport te creëren van de verschillende waarden. Voor het kwaliteitsdashboard is het meer nodig om een projectie te kunnen doen van de kwaliteitswaarden en dit één oogopslag.

6.4 Power BI

6.4.1 *Introductie*

Power BI is een *Business Intelligence* tool die gericht is naar het analyseren van bedrijfswaarden. De analyses worden opgebouwd uit dashboards die grafieken en diagrammen kunnen combineren uit verschillende datasets. De *dashboards* kunnen interactief gebruikt worden om specifieke waarden aan te tonen en helpen bij het creëren van een dieper inzicht. Het doel van Power BI is dat de dashboards organisaties kunnen helpen bij het nemen van beslissingen. Ook biedt Power BI de mogelijkheid om de *dashboards* online te gebruiken via een computer, tablet of smartphone. [20]

6.4.2 *Instapniveau*

Power BI is een *tool* die opgebouwd is net als de Office-programma's van Microsoft. Het werkt met veel knoppen die alle functionaliteiten moeten aanbieden. Indien er iets niet direct gevonden kan worden, is er een zoekfunctie die helpt bij het vinden er van. Alles is eenvoudig te interpreteren en vergt weinig tijd om te begrijpen hoe Power BI nu werkt. Ook Power BI biedt voldoende fora aan (zowel officiële forums van Microsoft als anderen) die helpen bij veel gestelde vragen en gevonden problemen.

6.4.3 *Voorbereiding*

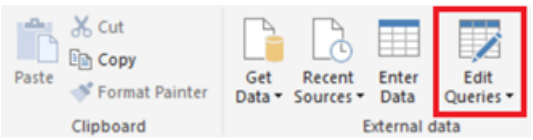
Power BI is een applicatie die voor gebruikers zowel een gratis als betalend pakket aanbiedt. Wanneer er met een gratis licentie van Power BI gewerkt wordt, zijn alle functionaliteiten in de applicatie voor de desktop beschikbaar. Indien er publicaties gemaakt moeten worden van de gemaakte *dashboards*, moet er een betalende licentie gebruikt worden. Voor dit onderzoek is er gebruik gemaakt van een betalende licentie.

Om te starten met Power BI moet de software lokaal geïnstalleerd worden. Daarna kiest de gebruiker de verschillende databronnen en kan er gestart worden met de visualisatie.

6.4.4 *Data Cleaning*

Power BI biedt de mogelijkheid om databronnen uit de lezen uit verschillende bestandtypes. Dat heeft als voordeel dat Microsoft het gemak aanbiedt om uit eender welke file data te laden en geautomatiseerd de structuur in een tabel te krijgen die direct bewerkbaar is.

Data bewerken in ingeladen tabellen gebeurt via *Edit Queries* in het tabblad van de databronnen.



Figuur 35: Bewerken van query's in Power BI.

Net als bij Pandas en Plot.ly moet de titel van de *build* opgesplitst worden om daaruit de datums van elke *build* te verkrijgen. Dat wordt gedaan zonder code te schrijven en is volledig doenbaar met behulp van de interface.

Een functionaliteit waar er input verwacht wordt van de gebruiker is wanneer er moet aangetoond worden op welk symbool er een kolom opgesplitst moet worden.

Waarden veranderen binnen een kolom wordt gedaan via de optie *Replace values*. Als input wordt meegegeven welke waarden veranderd moeten worden en waarin deze veranderd moeten worden.

Voor elke kolom in een tabel moet er meegegeven worden welk type van waarden er in opgeslagen is. Dat heeft als voordeel dat er automatische structuren toegepast worden die helpen voor een beter overzicht in de tabel. Daarnaast helpt de automatisering van dit ook bij het visualiseren. Als voorbeeld wordt getoond wat het effect is als een kolom *Date* omgezet wordt van *Text* naar *Date.*

Date	Date
2/4/2019	Monday, February 4, 2019
2/4/2019	Monday, February 4, 2019
2/5/2019	Tuesday, February 5, 2019
2/5/2019	Tuesday, February 5, 2019
2/5/2019	Tuesday, February 5, 2019
2/6/2019	Wednesday, February 6, 2019
2/7/2019	Thursday, February 7, 2019
2/7/2019	Thursday, February 7, 2019
2/8/2019	Friday, February 8, 2019

Figuur 36: Verwerken dan DateTime in Power BI.

Een nadeel wat hieraan vasthangt is dat voor bepaalde formaten (vb. *DateTime*) er expliciet aan het UTC-formaat voldaan moet worden. Er moet de nodige data *cleaning* gebeuren om alle data leesbaar te krijgen voor Power BI. Wanneer er tussen uren een "-" staat in plaats van ":", dan herkent Power BI dit niet als een *Time*-type.

Een nadeel van Power BI is dat het inladen van de data en het wijzigen van de data tijdrovend is. Bij elke update die moet gebeuren van de dataset, moet die volledig opnieuw ingeladen worden. Wanneer er gewerkt wordt met datasets die uit meer dan 100 000 rijen bestaat, kan het vanaf dan tot 10 minuten duren voordat er terug verder gewerkt kan worden. Hetzelfde geldt voor wanneer er een wijziging gebeurt met *data cleaning.*

6.4.5 *Updaten Datasets*

Het updaten van datasets gebeurt met de knop *Refresh*. Die controleert voor elke dataset die er ingeladen is in het project welke wijzigingen er gebeurt zijn. Zo wordt voor elke rij in elke *dataset* afgegaan of er een wijziging gebeurt is. Het gemak van Power BI zit in het eenvoudig updaten van de *datasets.* De tijd van het updaten van de *datasets* kan verschillen tussen een minuut tot 15 minuten afhankelijk van het aantal en de grootte van de *datasets*. Tijdens het eindwerk is er gewerkt geweest met 10 verschillende *datasets* waarvan sommigen tot 200.000 rijen bevatten. De laadtijden duren dan tot 15 minuten. Tijdens het updaten kan er niet verder gewerkt worden aan het project.

6.4.6 *Visualisatiemogelijkheden*

Het visualiseren van geselecteerde waarden kan gebeuren met behulp van de beschikbare mogelijkheden die terug te vinden zijn in het visualisatiemenu van Power BI. In tegenstelling met Jupyter Notebook is het een eenvoudig *drag-and-drop*-systeem waarbij elke optie op het dashboard geplaatst kan worden naar keuze. Daarna kan er onder *Values* de waarden uit de gebruikte databronnen geselecteerd worden. Als resultaat wordt de visualisatie automatisch veranderd afhankelijk van de ingegeven waarden.

Omgekeerd kunnen de waarden ook eerst geselecteerd worden en worden deze automatisch op het dashboard in een tabel weergegeven. Daarna kan de keuze gemaakt worden hoe deze gevisualiseerd moet worden.

Er is een uitgebreide verzameling aan tabellen, grafieken en diagrammen die gebruikt kunnen worden op het *dashboard*. Afhankelijk van de instellingen die de ontwerper van de visualisatie mogelijk gemaakt heeft, kan het wijzigen van de instellingen eerder beperkt zijn.

Indien er geen juiste visualisatieoptie standaard aanwezig is in Power BI, biedt een *marketplace* extra mogelijkheden aan.

6.4.7 *Live data visualiseren*

Een extra eigenschap van Power BI die Plot.ly niet heeft, is het verwerken van data die live moet gevisualiseerd worden. Om softwarekwaliteit te visualiseren is het bij bepaalde onderwerpen van belang op zo snel mogelijk output te zien van resultaten.

Om live data te visualiseren moet er met de online versie van Power BI gewerkt worden. Daarbij wordt in het project gewerkt met een *Streaming Dataset*. Die *Dataset* kan gebruikt worden door middel van een *API, Azure Stream* of via *Pubnub*. Omdat de databronnen binnen de organisatie intern bewaard worden, moet er met behulp van een API de data doorgevoerd worden naar het gebruikte *dasboard*.

In tegenstelling tot een *dashboard* dat geen *streaming dataset* gebruikt, is het aantal visualisatiemogelijkheden beperkt tot een lijngrafiek, staafdiagram en een kaart die de waarde op dat moment aantoont. De waarden op het dashboard zullen dan veranderen wanneer er wijzigingen gebeuren aan de *dataset*.

6.4.8 *Opbouw Dashboard*

Power BI is software die specifiek gericht is naar het opbouwen van *dashboards*. Het spreekt voor zich dat het opbouwen dan ook vlot verloopt. Er kan met behulp van het *drag-and-drop*-systeem grafieken en diagrammen geplaatst worden op het canvas waar de gebruiker het wenst te hebben. Er kunnen meerdere tabbladen aangemaakt worden voor verschillende overzichten of om een *drillthrough* te maken van het oorspronkelijke *dashboard*. Daarbij wordt er op een specifieke visualisatie ingezoomd door in te gaan op een ander dashboard met de waarden van de oorspronkelijke visualisatie. Het *dashboard* kan achteraf online gepubliceerd worden om het te kunnen delen met andere gebruikers. Het *dashboard* is daarna ook toegankelijk voor elk apparaat dat toegang heeft tot het internet (Computer, laptop, tablet en *smartphone*).

6.5 Azure DevOps

6.5.1 *Introductie*

Azure DevOps is een *developer* service van Microsoft die gebruikt wordt om teams te helpen bij het inplannen van werk, samenwerken in *code development* en het bouwen, testplannen opstellen en vele andere mogelijkheden. Azure DevOps is een *cloud* systeem dat een functie aanbiedt om *dashboards* op te bouwen op basis van de beschikbare data binnen DevOps. [21]

6.5.2 *Instapniveau*

Omdat Azure DevOps een *developer*platform is, is het belangrijk om te begrijpen welke processen achter softwareontwikkeling zitten. Daarnaast is het online platform erg uitgebreid en zijn er verschillende onderdelen met elk hun eigen functies. Vooraleer er gewerkt kan worden aan visualisaties binnen Azure DevOps, is het dus belangrijk om te begrijpen welke onderdelen er zijn, hoe elk onderdeel aan elkaar gelinkt is en welke functionaliteiten er aanwezig zijn. In tegenstelling tot Power BI is alles minder rechtstreeks duidelijk gemaakt en vergt het meer tijd om te begrijpen.

6.5.3 *Voorbereiding*

Azure DevOps is een platform dat een gratis als een betalende versie heeft. Een gratis versie van het platform biedt een beperkt aantal functionaliteiten aan voor een beperkt aantal gebruikers. Wanneer er overgegaan wordt naar de betalende versie van Azure DevOps, moet er maandelijks betaald worden op basis van het aantal gebruikers die toegang moeten krijgen tot het platform. Bij een betalende versie van Azure DevOps, zijn alle functionaliteiten toegankelijk ongeacht het aantal gebruikers.

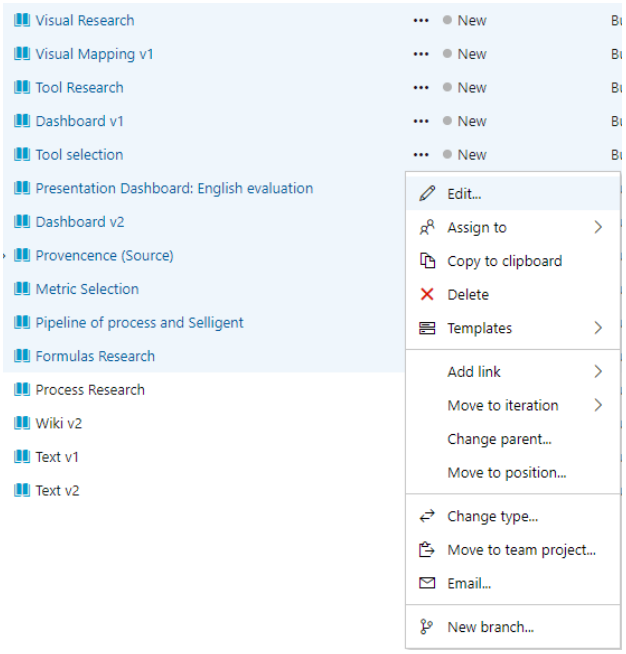
Om toegang te krijgen tot de projecten van de organisatie in DevOps, moet er een account voorzien worden met de gewenste permissies om data te kunnen opvragen en eventueel te bewerken.

6.5.4 *Data Cleaning*

Omdat Azure DevOps een belangrijke informatiebron is van de *backlog*gegevens, heeft data *cleaning* een impact op de *backlog items*. Zo zijn alle teams en andere stakeholders binnen *engineering* verbonden aan deze data. Daarnaast kan het overzicht en de structuur tussen alle *items* snel verdwijnen wanneer er zonder duidelijke afspraken wijzigingen gebeuren aan informatie in de werkvelden (niveau van ernst, uitleg van *work items*, ect.).

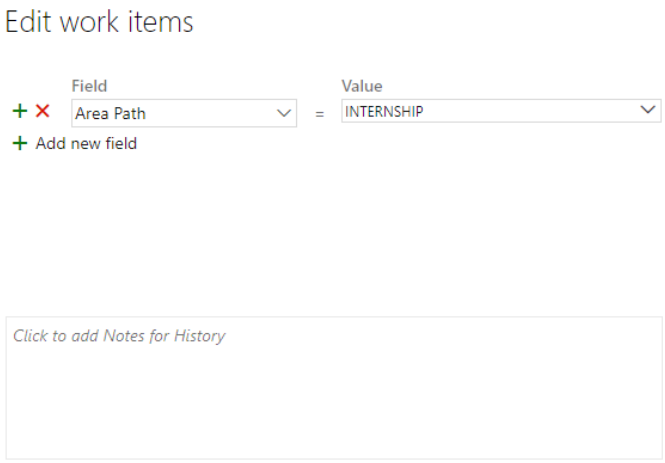
Data *cleaning* gebeurt op een manier waarbij de gegevens die gelinkt zijn aan items gewijzigd worden. M.a.w. De wijzigingen verwijderen oorspronkelijke waarden en kunnen niet meer terug opgevraagd worden.

Om aan *data cleaning* te doen op Azure DevOps, moet er via de *backlog* eerst gezocht worden naar de verzameling van *work items* waaraan wijzigingen doorgevoerd moeten worden. Dat wordt behaald door een *query* op te stellen en uit te voeren. Nadat de selectie van *items* gebeurt is, worden de items geselecteerd door de gebruiker en wordt de *Edit* functie opgeroepen. Op de volgende bladzijde wordt een schermafbeelding getoond van waar de functie zich bevindt.



Figuur 13: Edit functie Azure DevOps.

Waarden in specifieke velden van de geselecteerde *work items* worden bewerkt door het schrijven van een nieuwe query. Hierbij wordt ingegeven welke kolommen bewerkt moeten worden en welke waarde erin moet komen. Om documentatie van de wijziging bij te houden, is er een mogelijkheid om een notitie bij te houden in een apart tekstvak.



Figuur 14: Bewerken van geselecteerde waarden.

Nadat de wijziging doorgevoerd is, kan de nieuwe waarde gezien worden door individuele items te controleren in een apart venster. Een andere mogelijkheid is om met een nieuwe zoekquery naar de nieuwe waarde te zoeken. Indien er in de notities informatie ingegeven is omtrent de wijziging van waarden, kan deze teruggevonden worden als opmerking onderaan elk *work item* venster.

6.5.5 *Updaten Datasets*

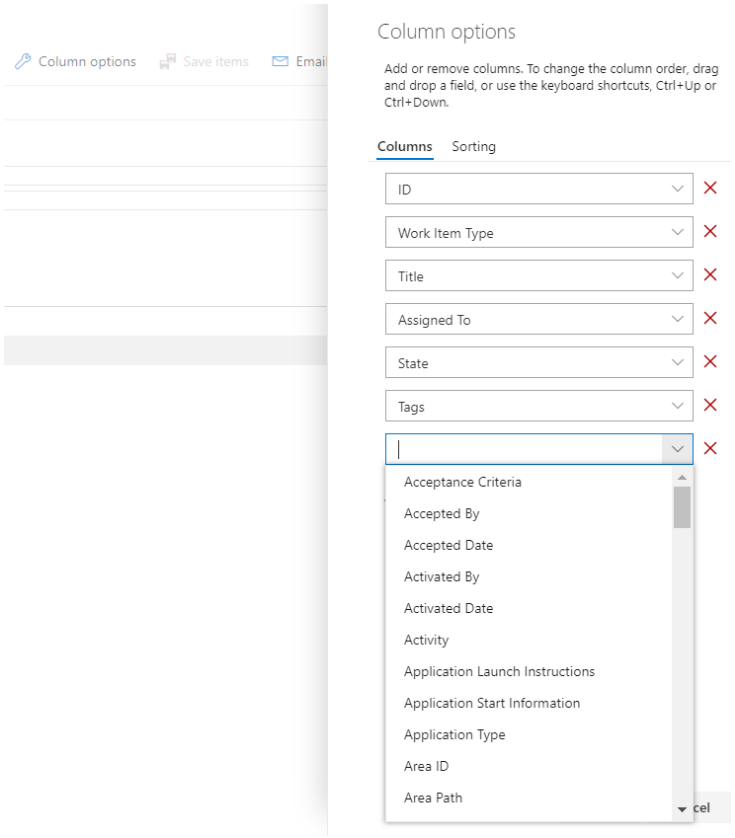
Omdat Azure DevOps constant de geschreven query's bijwerkt en de gebruikte *datasets* rechtstreeks gelinkt zijn aan de *backlog*, is het updaten dus een eigenschap die constant gebruikt wordt.

6.5.6 *Live data*

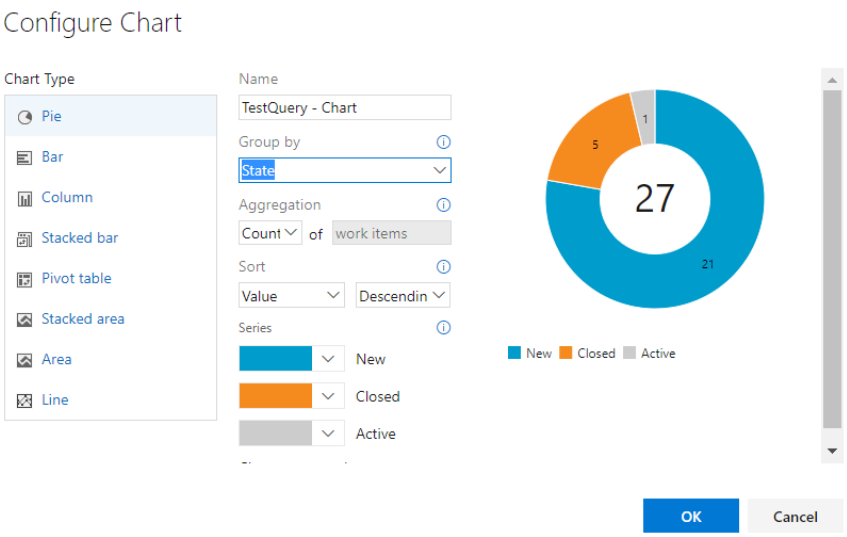
Omdat Azure DevOps een databron is die gebruikt wordt om alles rond de softwareproductie bij te houden, is er het voordeel dat wanneer er *dashboards* opgebouwd worden, deze vasthangen aan query's die constant geüpdatet worden wanneer er een wijziging gebeurt aan de *backlog*. Afhankelijk van hoe belastend de query is, kan er een kleine *delay* zijn bij het inladen van de output. De query's die het meest belastend zijn om *items* uit de *backlog* te halen duurt maximaal 10 seconden. Het updaten van *datasets* in Power BI kan tot 10 minuten duren, afhankelijk van de gevraagde *work items*.

6.5.7 *Gebruik van Azure DevOps*

In tegenstelling tot Power BI is er bij Azure DevOps geen extra stap nodig om de data binnen te halen om er visualisaties mee te kunnen maken. Omdat Azure DevOps een bron is waar data al reeds opgeslagen wordt, kan er dus rechtstreeks gewerkt worden met de beschikbare data. Om de data te gebruiken wordt er met query's gewerkt die opgeslagen worden in het systeem. In het querymenu is er een tabblad *Charts* waarin er visualisaties opgebouwd kunnen worden en daarna in een *dashboard* geplaatst worden. Om bepaalde kolommen uit de query te gebruiken moeten deze in de opties geselecteerd worden vooraleer deze beschikbaar zijn voor een visualisatie.



Figuur 15:Toevoegen van kolommen in query.



Figuur 16:Visualisatiemogelijkheden in Azure DevOps.

6.5.8 *Visualisatiemogelijkheden*

Voor de visualisatie van de query is er een aanbod aan mogelijkheden. In vergelijking met Power BI is dit veel minder. Echter is er ook voor het opbouwen van een *dashboard* in Azure DevOps een *marketplace* waar er verschillende andere mogelijkheden zijn. Echter zijn er bepaalde opties betalend en moet er ook een permissie zijn om deze te gebruiken door een geautoriseerd persoon.

6.5.9 *Opbouw Dashboard*

Een *dashboard* in Azure DevOps wordt opgebouwd in een raster. Er worden grafieken en diagrammen in deze rasters geplaatst en de grootte kan gewijzigd worden in de opties van de visualisatie. Wat onmiddellijk opvalt is dat er geen vrijheid is in het plaatsen van de blokken. Er kunnen geen *items* apart geplaatst worden en alles wordt automatisch onder elkaar geplaatst. Dat kan storend zijn omdat er geen ruimte is voor open plekken. In Power BI is dit een voordeel om alles te kunnen plaatsen waar de gebruiker het wilt.

6.6 Toolvergelijking

Om de drie tools te vergelijken met elkaar, kan op basis van 7 eigenschappen onderstaande tabel opgesteld worden.

	Jupyter Notebook	Azure DevOps	Power BI
Instapniveau	✦	✦✦	✦✦✦
Voorbereiding	✦	✦✦	✦✦✦
Data *Cleaning*	✦✦✦	✦	✦✦
Updaten *Datasets*	✦✦	✦✦✦	✦
Live Data	✦	✦✦✦	✦✦
Visualisatiemogelijkheden	✦✦✦	✦	✦✦
Opbouw Dashboards	✦	✦✦	✦✦✦

Tabel 1: Toolvergelijking.

Jupyter Notebook en de Python *libraries* zijn zeer sterke tools die gebruikt kunnen worden om uitgebreid en verdiepend data in kaart te brengen en meerwaarde te geven. Eens de *syntax* en de manier van werken gekend zijn kan er op een zeer snelle manier inzicht gegeven worden. Echter is het zeer omslachtig om data continue te updaten. Er komt veel manueel werk bij kijken vooraleer de data-interpretatie kan gebeuren. Plot.ly heeft de sterkte dat het naast lokaal gebruik via Jupyter ook op een online platform herbekeken kan worden.

Elke visualisatie wordt op het account bijgehouden ongeacht of het wordt verwijderd uit het rapport van Jupyter Notebook. Bij Pandas is het instapniveau om aan visualisaties iets hoger omdat de syntax complexer is om een grafiek of diagram op te stellen. Echter is het bewerken van de data veel eenvoudiger en is de output van de wijzigingen met meer structuur gevisualiseerd.

Azure DevOps is een uitstekende tool om basis visualisaties op te bouwen. De sterkte van dit platform is dat het naast een bron van data ook gebruikt kan worden om deze data in kaart te brengen. Ook het constant updaten van query's wanneer er veranderingen gebeuren is een voordeel tegenover Plot.ly en Power BI.

Het nadeel is echter dat er enkel gewerkt kan worden met data die enkel beschikbaar is op Azure DevOps.

Omdat de resultaten van automatische testen niet toegankelijk zijn in dit platform en dit gezien wordt als een belangrijke kwaliteitswaarde is, is Azure DevOps niet geschikt voor het doel van dit onderzoek.

Power BI is een tool die weinig introductie nodig heeft en is eenvoudig om aan te leren aan personen die geen ervaring hebben met het visualiseren van data. Het *drag-and-drop*-systeem maakt het erg eenvoudig en snel om een *dashboard* op te bouwen. Daarnaast is er veel documentatie beschikbaar over veelgestelde vragen. Voor data *cleaning* is het ook erg eenvoudig om wijzigingen door te voeren. Er kunnen eigen query's geschreven worden voor een uitgebreidere wijzigingen, maar ook zijn er visuele knoppen en opties om *datasets* te veranderen.

Het nadeel dat voorkomt bij Power BI is de beperktheid in de opties van de grafieken en tabellen. Er zijn veel mogelijkheden om uit te kiezen, maar weinig opties zijn uitgebreid aan te passen. Daarnaast is het updaten van *datasets* erg tijdrovend doordat er over elke rij heengegaan wordt of er iets gewijzigd is. De laadtijden zijn afhankelijk van de grootte en het aantal *datasets.*

Concreet kan gesteld worden dat wanneer er uitgebreid en verdiepend gewerkt moet worden met data, Jupyter Notebook met Pandas of Plot.ly een geschikte combinatie is om mee aan de slag te gaan. Wanneer er een snel inzicht gemaakt moet worden zonder er specifieke eisen zijn van hoe de visualisaties er uitzien, is de *dashboard* optie binnen Azure DevOps voldoende. Power BI is geschikt wanneer er een combinatie gevraagd is van gebruiksvriendelijkheid, snel opbouwen van visualisaties en toch uitgebreide opties hebben om extra inzichten te krijgen.

Omdat er een mogelijkheid is bij Power BI om een link te leggen met het Azure DevOps platform en dit te combineren valt met andere databronnen, wordt Power BI gebruikt om het kwaliteitsdashboard op te bouwen. Ook de iets uitgebreidere keuzes om aan data *cleaning* te doen en de bredere keuze aan diagrammen en grafieken, zijn redenen om Power BI te gebruiken.

Er kan als conclusie gesteld worden dat er geen tool beter is als de ander, de keuze moet gemaakt worden op basis van wat de eindgebruik als eindresultaat wilt hebben. Indien snelle inzichten in een rapportvorm nodig zijn met een uitgebreide mogelijkheid om de visualisaties te personaliseren, dan is Jupyter Notebook geschikt. Indien er met live data gewerkt moet worden en een *real time*status belangrijk is, dan is Azure DevOps geschikt genoeg met zijn visualisatietool. Als er een combinatie van beide nodig is waarbij er eenvoudig gewerkt moet worden met data en toch een uitgebreide keuze moet zijn om te visualiseren, dan is Power BI geschikt.

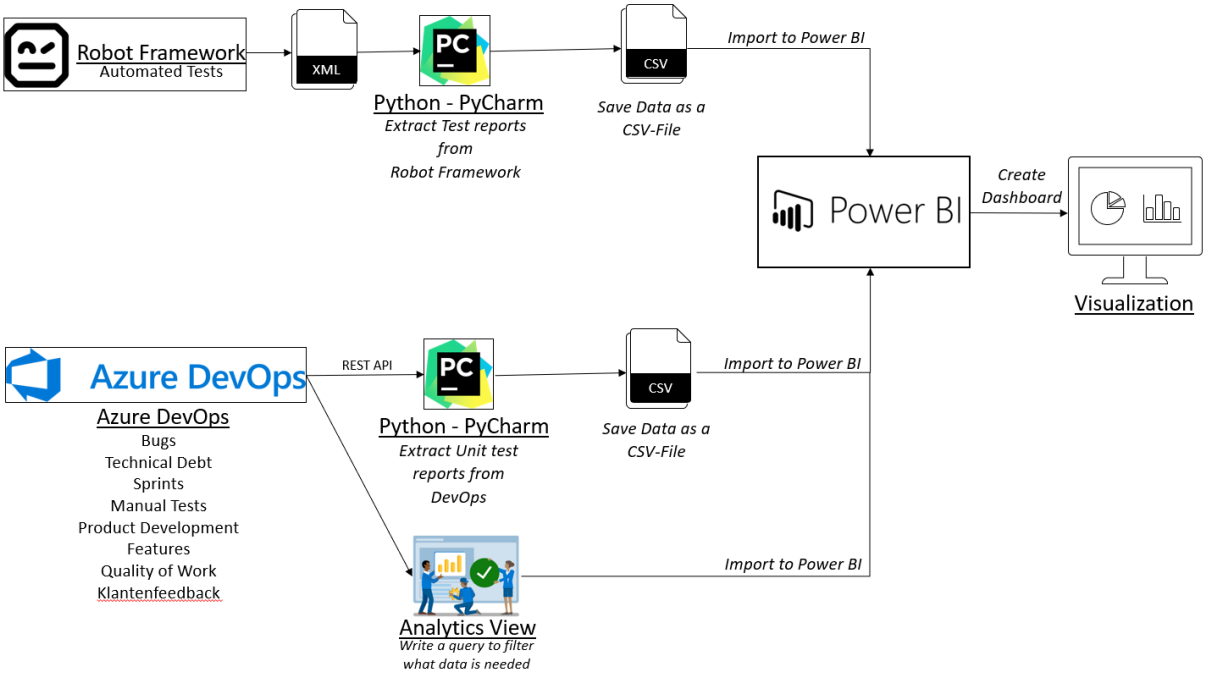
7 Visualisaties

7.1 Kwaliteitsdashboard

7.1.1 *Inleiding*

Als besluit wordt gesteld dat Power BI de visualisatietool is waarmee het kwaliteits*dashboard* wordt opgebouwd. Als bekend is welke kwaliteitswaarden in kaart worden gebracht, wordt de *mockup* versie van het uiteindelijke dashboard uitgetekend. Voor dit onderzoek wordt een *dashboard* opgebouwd dat specifiek gericht is naar de productkwaliteit. Daarnaast wordt er verder ook een proceskwaliteitsdashboard opgesteld verder in het onderzoek. Deze samen geven een geschikt beeld van de softwarekwaliteit.

7.1.2 *Databron Diagram*



Figuur 17: Databron diagram van het kwaliteitsdashboard.

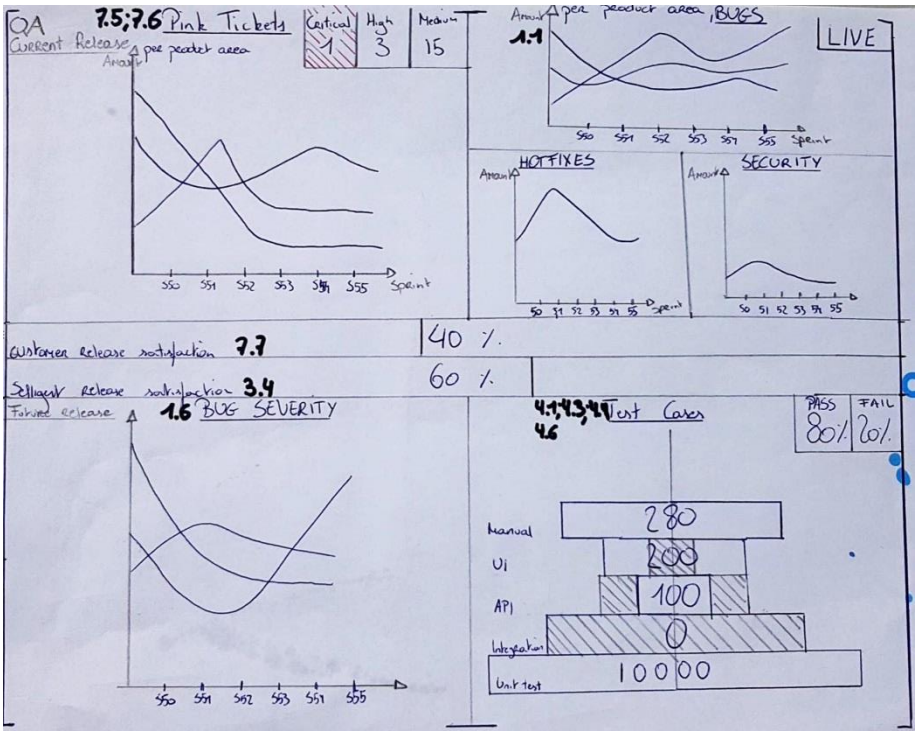
Het kwaliteitsdashboard wordt opgebouwd door data te verkrijgen van het Robot Framework om de automatische testresultaten te krijgen en Azure DevOps om de andere kwaliteitswaarden te visualiseren. Als uitgelegd in hoofdstuk *5 Databronnen*. Wordt uitgelegd hoe de data opgehaald wordt via deze twee databronnen.

Omdat er op het kwaliteitsdashboard een visualisatie gebeurt van het aantal *unit* testen en deze niet via *analytics view* toegankelijk is, wordt via de REST API van Azure DevOps deze data opgehaald via een geschreven Python-script. Alle andere gegevens worden via *analytics view* opgehaald om daarna in Power BI te verwerken.

7.1.3 *Mockup*

Selectie van kwaliteitswaarden

Om de kwaliteit te bepalen van de software moet er een visueel beeld geschept worden van drie onderdelen die rechtstreeks hieraan gelinkt zijn. Een eerste onderwerp is het visualiseren van de bugs. Daarbij moet er gekeken worden naar hoeveel bugs er op het huidige moment aanwezig zijn en wat de ernst van deze bugs zijn. Daarnaast is het van belang om te weten welke bugs gelinkt zijn aan *pink tickets*. Aangezien klanten de eindgebruikers zijn van het product, moet het gekend zijn welke problemen zij ondervinden. Als laatste is het voor softwarekwaliteit van belang om het effect van de verschillende testfasen te visualiseren. Verder in dit onderdeel wordt dieper ingegaan op de verschillende onderdelen die te zien zijn in de *mockup*.

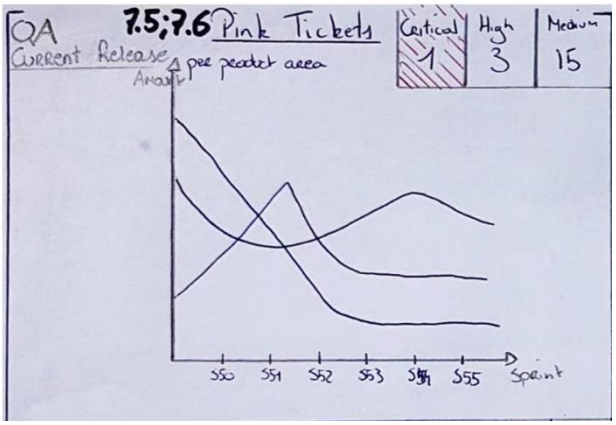


Figuur 18: Mockup dashboard softwarekwaliteit. (Dummy Data)

Bugs gerelateerd aan Pink Tickets

Binnen de organisatie wordt de focus gelegd op hoe elke release verloopt. Nadat er een nieuwe release bij de klanten terechtkomt, moet er vanaf dan bijgehouden worden in welke onderdelen van het product er bugs optreden. Aan de hand van een lijngrafiek wordt er gevisualiseerd hoeveel bugs er per productgebied gemeld worden. Dat wordt gedaan door de releaseperioden te gebruiken als tijdsperioden.

Daarnaast wordt met behulp van drie kaarten een indicatie gegeven over de ernst van de bugs die gerelateerd zijn aan *pink tickets.* Voor de types *Critical, High* en *Medium* wordt bijgehouden hoeveel actieve bugs er aanwezig zijn met deze waarde van ernst.



Figuur 19: Bugs gelinkt aan pink tickets. (Dummy Data)

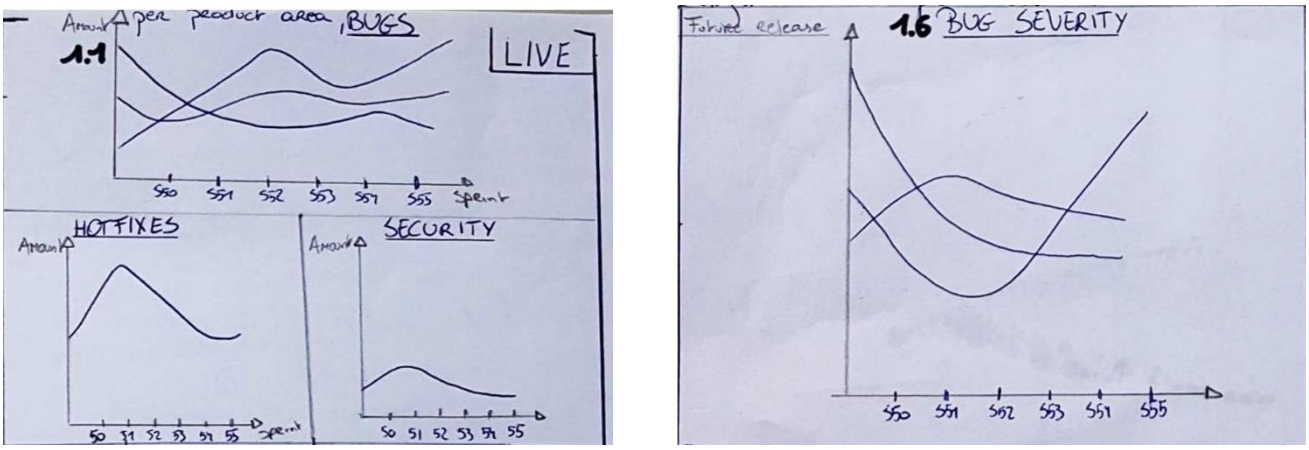
Softwarebugs

Naast het oplossen van bugs die gevonden worden door klanten, moet er ook een overzicht zijn van het aantal actieve bugs die er binnen de organisatie gevonden worden. In het *dashboard* moet er verwerkt worden hoeveel bugs er actief zijn per productgebied. Dat wordt net als bij kwaliteitswaarde *Bugs density* gedaan door een lijngrafiek te gebruiken waarbij er per release aangetoond wordt hoeveel bugs er zijn per productgebied.

De kwetsbaarheid van software is een aspect wat ook in kaart gebracht moet worden tijdens de productie. Zo wordt in het dashboard een lijngrafiek voorzien die in kaart brengt hoeveel bugs er actief zijn die gerelateerd zijn aan beveiligingsproblemen.

Ook is er een onderdeel op het *dashboard* beschikbaar dat dient om het aantal *hotfixen* per release te tonen. Door deze waarde in kaart te brengen kan er geëvalueerd worden hoeveel bugs een *hotfix* krijgen tijdens elke release. Op de volgende bladzijde wordt het onderdeel op het *dashboard* individueel getoond.

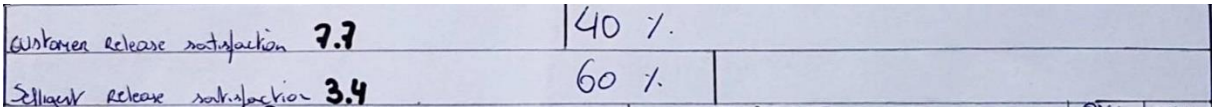
Als laatste wordt een plek voorzien waarbij het aantal bugs die actief zijn worden gesorteerd per ernstniveau. Ook bij deze waarde wordt per release vergeleken.



Figuur 20: Onderdeel bugs en hotfixen (links) en bugs gesorteerd per ernst (rechts). (Dummy Data)

Releasetevredenheid

Om te begrijpen hoe er intern en extern gedacht wordt over een laatste release, wordt op het *dashboard* een vergelijking voorzien van de tevredenheid. Twee staafdiagrammen tonen in vergelijking met elkaar hoe de laatste release was voor de klanten en voor het personeel van de organisatie.



Figuur 21: Tevredenheid van laatste release. (Dummy Data)

7.1.4 *Uitwerking kwaliteitsdashboard*

Op basis van de getekende *mockup* wordt er afgeleid dat de geschikte tool om dit te realiseren Power BI is. Dat omwille van de gebruiksvriendelijkheid om *dashboards* op te bouwen, de mogelijkheid om verschillende *datasets* te combineren met elkaar en voornamelijk de interactieve opties die kunnen dienen om extra inzicht te krijgen op specifieke waarden. De andere besproken tools (Jupyter Notebooks en Azure DevOps) bezitten deze functionaliteiten minder tot helemaal niet.

Gebaseerd op de getekende *mockup* is er in Power BI een *dashboard* opgebouwd die deze waarden moeten visualiseren. De reeds beschreven databronnen worden in dit dashboard gelinkt en worden in verschillende onderdelen gebruikt. Verder in dit hoofdstuk wordt per onderdeel besproken welke data uitgelezen kan worden en hoe deze geïnterpreteerd kan worden.

Elk vak in het dashboard wordt individueel gebruikt. Er wordt dus geen correlatie gemaakt tussen de verschillende onderdelen.

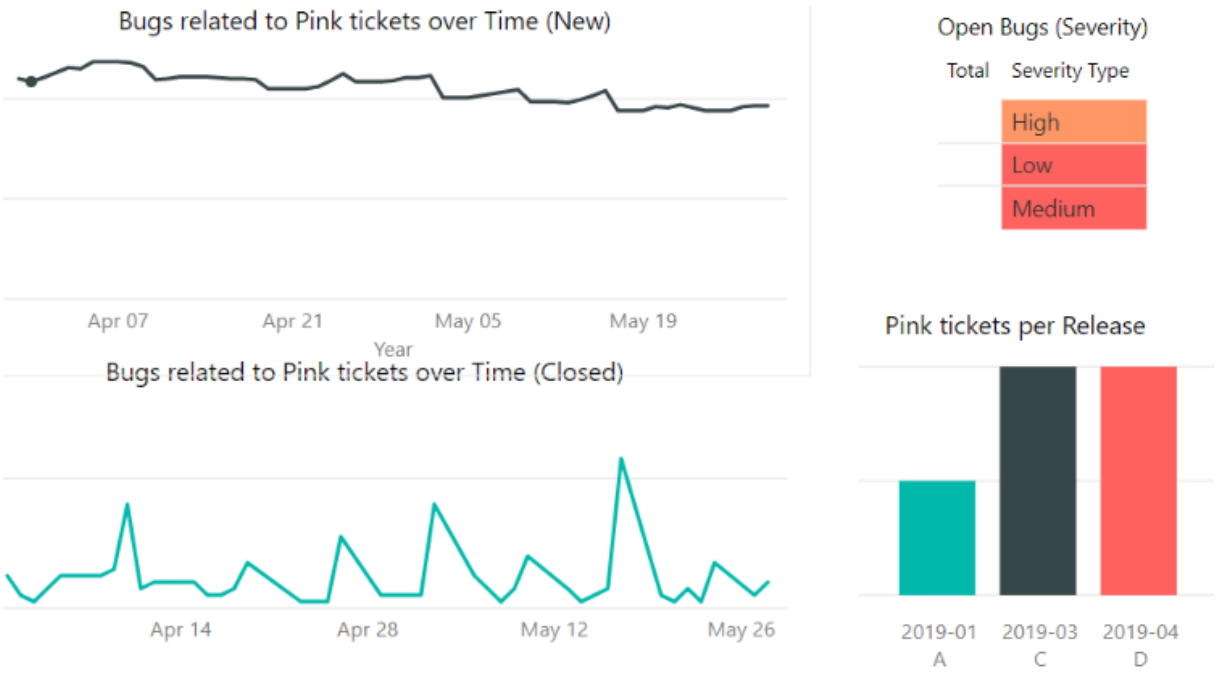
Het onderdeel waarin de tevredenheid van klanten en teams binnen de organisatie zijn uit de scope van het *dashboard* geplaatst. Na overleg met stakeholders binnen *Sales* blijkt de tevredenheid van klanten te complex te zijn om in één cijfer te visualiseren. Voor deze waarde moet er een alternatief *dashboard* opgebouwd worden om dit beter in kaart te brengen.

Waarden rond productonderdelen en releasenamen zijn uit de afbeeldingen verwijdert als bescherming van de interne data van Selligent.

Om interne data te beschermen zijn specifieke gegevens verwijdert uit de *screenshots* van het dashboard.

Figuur 22: Kwaliteitsdashboard uitgewerkt in Power BI.

7.1.5 *Bug gerelateerd aan Pink Tickets*



Figuur 23: Bugs gelinkt aan pink tickets.

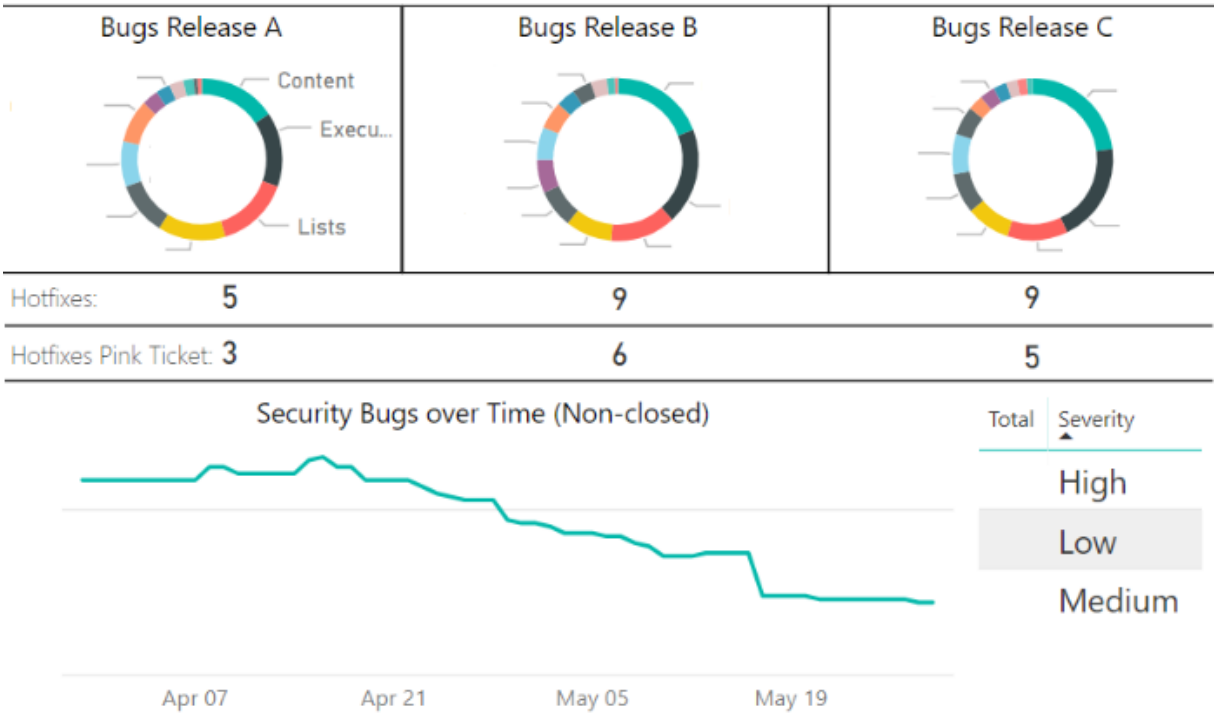
Links bovenaan wordt een verdeling gemaakt tussen verschillende kwaliteitswaarden. Als eerste wordt een lijngrafiek weergegeven die aantoont hoeveel klantgerelateerde bugs er met staat *New* (dus niet opgelost) nog in de *backlog* aanwezig zijn. Met behulp van de tabel (rechts), wordt een mogelijkheid aangeboden om per niveau van ernst te kijken hoeveel bugs er doorheen te tijd meer of minder bijkomen. Elk ernstniveau krijgt een andere kleur afhankelijk van de hoeveelheid bugs die gelinkt zijn aan dat niveau.

Naast de nieuwe bugs is er ook een inzicht in het aantal klantgerelateerde bugs die per dag afgerond worden. Zo kan er vergeleken worden met het nieuwe aantal bugs of er meer *pink tickets* binnen komen dan dat er opgelost worden.

Als laatste is er een staafdiagram die per *release* van SMC het aantal bugs toont die klantgerelateerd zijn. Binnen Selligent is het belangrijk om hierin een inzicht te krijgen omdat het systeem van *releases* vrij nieuw is en dus goed opgevolgd moet worden. Dat wordt gedaan om te kunnen streven naar *continuous delivery*.

De data die gebruikt wordt voor dit onderdeel komt rechtstreeks uit de *backlog* bewaard in Azure DevOps. Met behulp van *analytics view* wordt de data als een dataset gevormd om in het *dashboard* te gebruiken.

7.1.6 *Releasewaarden*



Figuur 24: Releasewaarden.

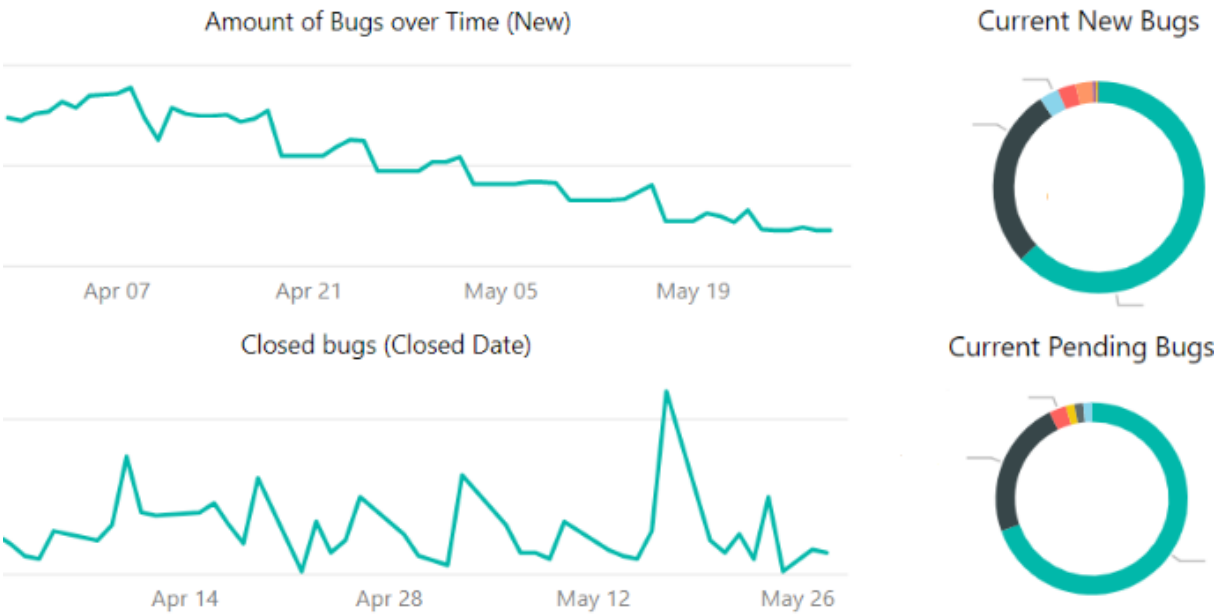
Omdat het in kaart brengen van de evolutie per release van hoog belang is, wordt een onderscheid gemaakt tussen het aantal bugs die per release gevonden worden. Zo kan het aantal telkens afgelezen worden uit het midden van het cirkeldiagram. In elk cirkeldiagram wordt de verhouding van het aantal *bugs* per product onderdeel gevisualiseerd.

In vergelijking met de *mockup* worden de *hotfixes* met een waarde onder elk cirkeldiagram aangetoond. Indien er geen *hotfixes* zijn voor een bepaalde *release*, wordt deze als '(Blank)' getoond. De verandering tegenover de *mockup* is gemaakt om een het verschil tussen releases eenvoudig zichtbaar te maken.

Onder deze waarden wordt in een lijngrafiek getoond hoeveel *bugs* er aanwezig zijn doorheen de tijd. Wanneer er *bugs* de staat 'Closed' krijgen, worden deze niet meer meegeteld in deze grafiek. *Bugs* gerelateerd aan *security* worden niet rechtstreeks in verband gelegd met *releases*. Bij een lijngrafiek per *release* zou er op de X-as dus meer 'Blank' staan dan effectieve releasenamen.

Ook hier zijn alle waarden gevisualiseerd met behulp van *analytics view* via Azure DevOps.

7.1.7 *Bugs aanwezig in de Software*

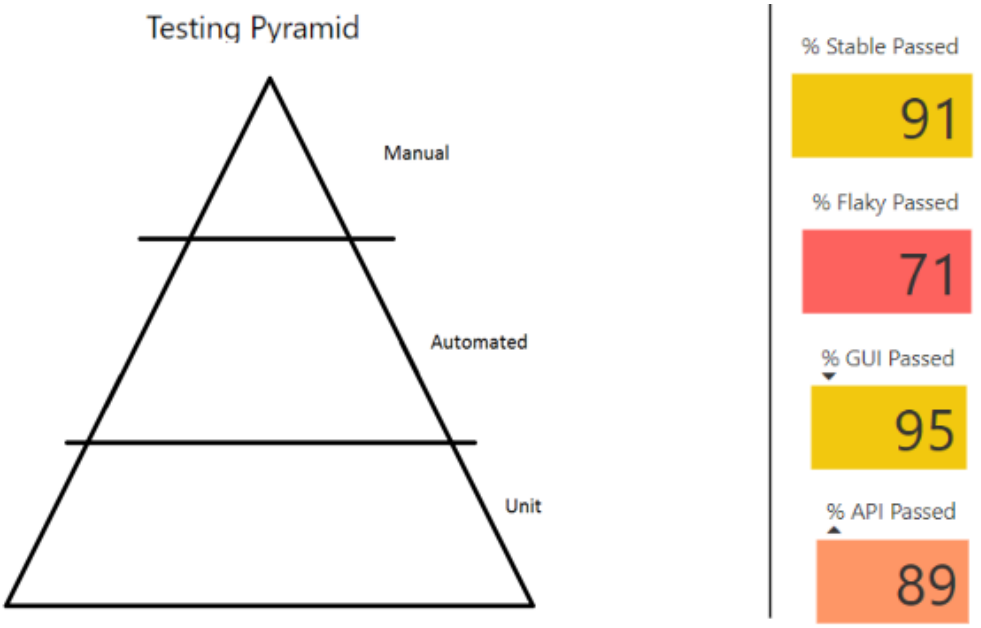


Figuur 25: Aantal bugs actief doorheen de tijd.

Om de aanwezigheid van bugs (intern en door klanten gevonden) op te volgen, is er links onderaan een sectie voorbehouden. Net als bij het onderdeel *pink tickets*. Is er bovenaan een grafiek te zien die het aantal bugs doorheen de tijd toont die nog als *New* in de backlog staan. Onder deze lijngrafiek is er ook weer te zien hoeveel bugs er dagelijks met succes gemaakt worden (en dus als status *Closed* krijgen). Het bovenste cirkeldiagram geeft het exacte aantal actieve bugs weer met een onderverdeling per productonderdeel. Het onderste cirkeldiagram toont het exacte aantal bugs waar aan gewerkt wordt of die gecontroleerd moeten worden of ze opgelost zijn. Ook hier is er een onderverdeling gemaakt per productonderdeel.

Data die gebruikt wordt in dit onderdeel komt uit Azure DevOps en is via een *analytics view* opgevraagd.

7.1.8 *Test Pyramide en resultaten Robot Framework*



Figuur 26:Testpyramide (Links) en resultaten Robot Framework (rechts).

Voor elke testfase die aanwezig is binnen de organisatie wordt met behulp van een trechterdiagram aangetoond hoeveel dit er zijn in verhouding met elkaar. Het doel dat bereikt moet worden op lange termijn is om al deze testfasen in de vorm van een pyramide te krijgen. Het aantal unit testen wordt gesplitst in JUnit testen en het aantal NUnit testen. Zo is het streefdoel om zoveel mogelijk geautomatiseerde testen te hebben en zo min mogelijk manuele testen. De reden om de focus te verschuiven naar automatische testen is omdat de *return on investment* bij automatische testen veel groter is in vergelijking met manuele testen.

Rechts kan er telkens de resultaten van de meest recente *run* van de automatische testen afgelezen worden. De afsplitsing wordt net als in het Robot Framework gemaakt tussen GUI en API testen. Net als bij het onderdeel van de *bugs* gerelateerd aan *pink tickets*, wordt op basis van kleurveranderingen aangetoond hoe goed of slecht het percentage is.

Daarbij is er ook een cijfer te zien van de *Stable* en *Flaky* testen. Elk kader verandert van kleur naarmate het slaagpercentage zakt. De kleuren veranderen van groen (=100%) naar geel (99% - 90%) naar oranje (89% - 80%) en uiteindelijk naar rood (<80%). Een uitgebreider rapport van *stable* en *flaky* testen kan teruggevonden worden onder *7.4 Rapport automatische testen.*

Voor de testpyramide worden meerdere databronnen gebruikt. Zo wordt het aantal automatische testen geteld door uit het laatste rapport uit te lezen. Het rapport bestaat uit een XML-bestand en het aantal keren dat het element *test* voorkomt wordt opgeteld.

Voor het aantal *unit* testen wordt met behulp van de REST API in een laatste *build* die gerelateerd is aan de *masterbranch* geteld hoeveel JUnit testen er aanwezig zijn. Die waarde wordt opgeteld met het aantal NUnit testen in diezelfde *build*.

Voor het aantal manuele testen is er met *analytics view* binnen Azure DevOps gewerkt.

Voor personen die verantwoordelijk zijn voor de softwarekwaliteit, is dit onderdeel van het dashboard van belang om een idee te krijgen van het werk dat verricht moet worden om het streefdoel van de pyramide te behalen.

7.1.9 *Interpretatie van het dashboard*

De Interpretatie van het dashboard is uit dit document gehaald om interne data te beschermen.

7.1.10 *Verdere optimalisatie*

Omdat de tijd die gespendeerd wordt aan dit eindwerk, is er veel plaats voor verbetering om het *dashboard* beter uit te werken. Zo zijn er problemen die op lange termijn de bruikbaarheid van het *dashboard* kunnen belemmeren.

Het huidige kwaliteitsdashboard vergt verschillende manuele updates van *datasets* om het *dashboard up-to-date* te houden met de huidige situatie van de kwaliteit binnen Selligent. Zo wordt voornamelijk tijd gespendeerd aan het automatiseren van het ophalen van resultaten van het Robot Framework.

Daarnaast bestaan de *datasets* die opgehaald worden via *analytics view* van Azure DevOps soms uit 100.000+ rijen. Dat zorgt er voor dat de laadtijden om het *dashboard* te updaten tot 15 minuten kan duren. Als oplossing om de laadtijden te optimaliseren moeten *datasets* in kleinere stukken verdeeld worden. Daarnaast moet er specifieker gekeken worden naar welke waarden gebruikt worden op het dashboard en welke data ingeladen wordt maar niet gebruikt wordt. Een laatste oplossing om de laadtijden te optimaliseren is door een beter datamodel op te stellen die gebruik maakt van elementen uit verschillende *datasets.*

Een laatste probleem is het manueel updaten van de cirkeldiagrammen per release. Op het huidige dashboard worden de releases manueel geselecteerd via het veld *Release name*. Indien er telkens voor de laatste drie *releases* een overzicht gegeven moet worden, moet er een alternatieve visualisatie gevonden worden om het *release*overzicht te verkrijgen.

Het updaten van het gehele *dashboard* gebeurt telkens manueel door de *Refresh*-knop te gebruiken. Ook dit moet geautomatiseerd worden om het dashboard zo bruikbaar mogelijk te maken.

7.1.11 *Conclusie productkwaliteit*

De Interpretatie van het dashboard is uit dit document gehaald om interne data te beschermen.

7.2 Team Dashboard

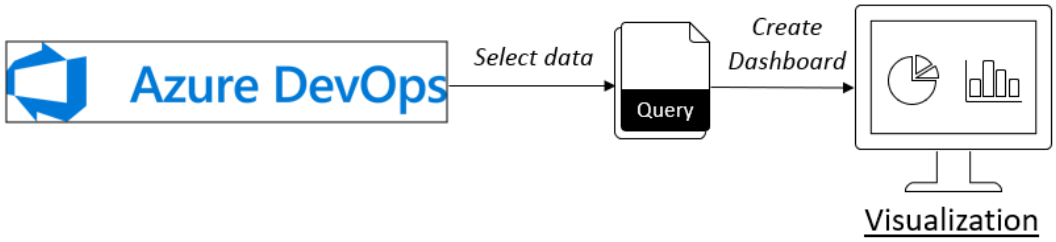
7.2.1 *Inleiding*

Binnen Selligent wilt men per team een universeel *dashboard* hebben waarop data gevisualiseerd wordt die in teken staat van een sprintplanning en review. Het *dashboard* is opgebouwd in de *dashboard*functie van Azure DevOps. Dat omdat het minder van belang is om uitgebreide visualisaties te kunnen doen en voornamelijk cijfergegevens van belang zijn. Daarnaast is het ook belangrijk om data te hebben die constant up-to-date is met de situatie. Omdat het dashboard uitgebreid is en niet op één scherm duidelijk zichtbaar is, wordt het dashboard per onderdeel uitgelegd.

Elke *widget* die in het *dashboard* gebruikt wordt is interactief met als nut om te kijken naar de query die gebruikt wordt om de visualisatie waar te maken. Dat kan gebruikt worden om specifieke informatie van de visualisatie op te vragen.

De kleuren die gebruikt worden voor elke kaart met een cijfer op en lijngrafieken zijn gerelateerd aan de kleur van elk *work item* type. In de *backlog* heeft elk *work item* type een eigen kleur. Bijvoorbeeld: bugs hebben een licht blauwe kleur en features worden in het paars weergegeven.

7.2.2 *Databron diagram*

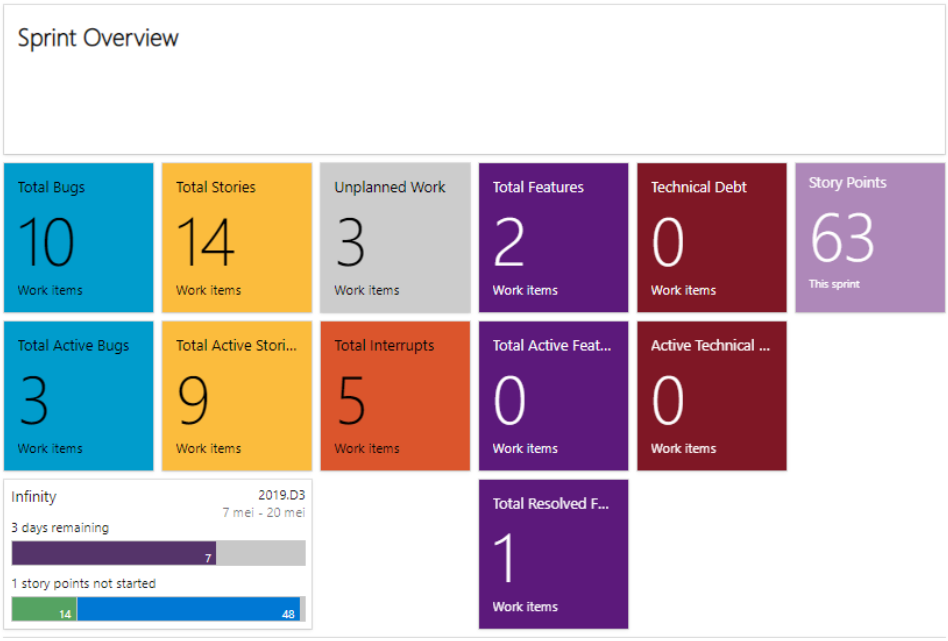


Figuur 27:Databrondiagram van het teamdashboard.

Om het teamdashboard op te bouwen wordt rechtstreeks data gebruikt uit Azure DevOps met de functionaliteiten van het platform. Azure DevOps werkt met *widget*-functies die query's gebruiken om data te visualiseren. De query's worden geschreven op basis van welke kwaliteitswaarde bijgehouden moet worden en de juiste data te selecteren.

7.2.3 *Uitwerking Dashboard*

Sprint Overzicht:

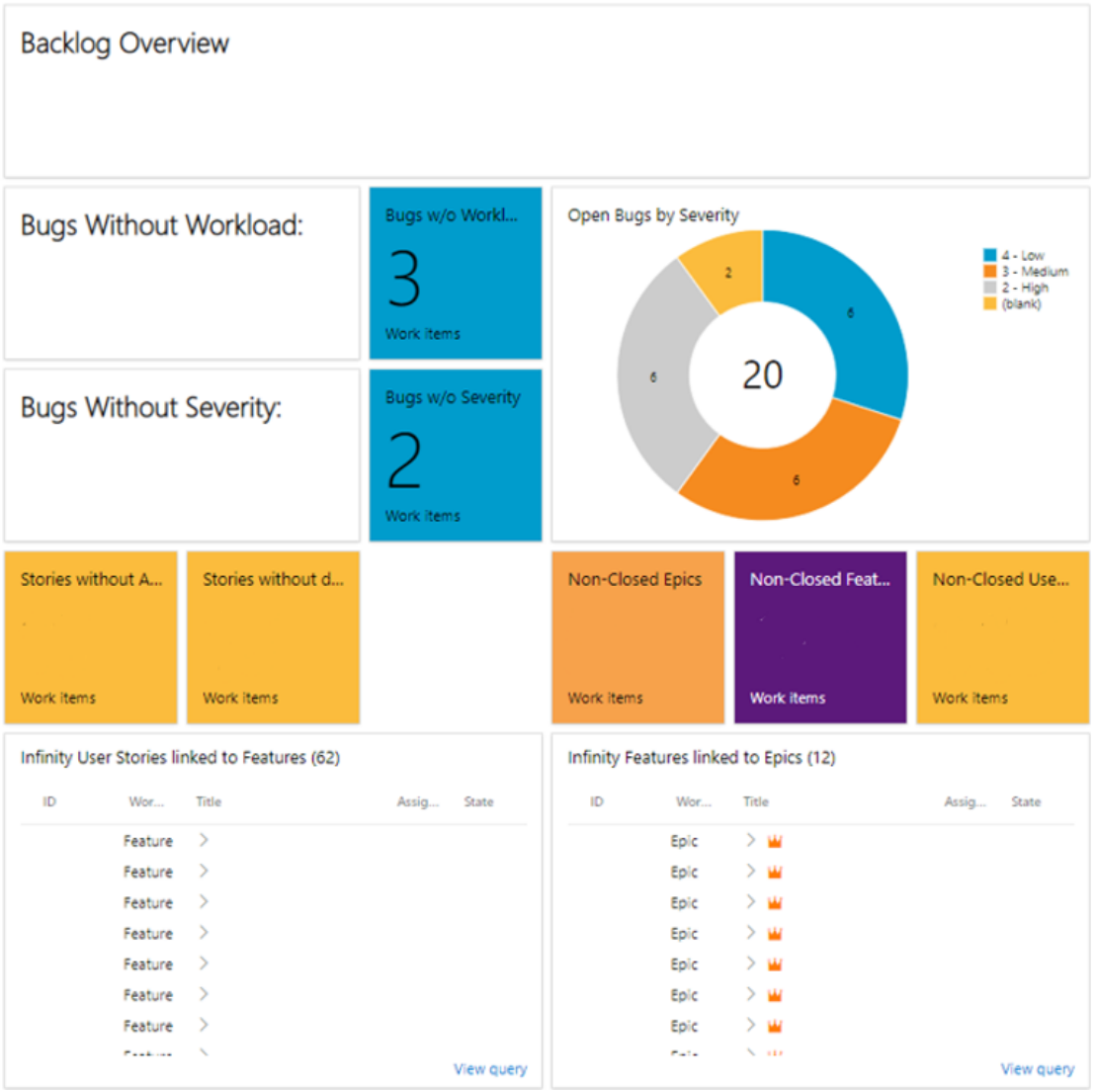


Figuur 28: Sprint overzicht van het team dashboard voor één team.

Bovenaan het *dashboard* is er een algemeen overzicht te zien waarop gegevens staan die informatie geven over de huidige sprint. *Work items* van het type bugs, *user story's,* reactief werk, *interrupts, features, technical debt* en *story points* van de huidige sprint worden in aantal getoond dat ze ingepland zijn en in hoeveelheid waar er actief aan gewerkt wordt. Dat biedt de mogelijkheid om in te kijken in hoeveel werk er in een team verricht wordt.

Naast de cijfers van de *work items* wordt een *widget* gebruikt die toont hoeveel geplande *story points* er gestart zijn om aan te werken (aangeduid in het groen) en hoeveel er afgerond zijn (aangeduid in het blauw). Daarbij wordt het aantal dagen dat een sprint nog duurt weergegeven. Dat onderdeel helpt het team om in te schatten hoeveel werk ze nog moeten afwerken om de sprint succesvol af te ronden.

Backlog Overview:



Figuur 29: Backlog overzicht van het team dashboard voor één team.

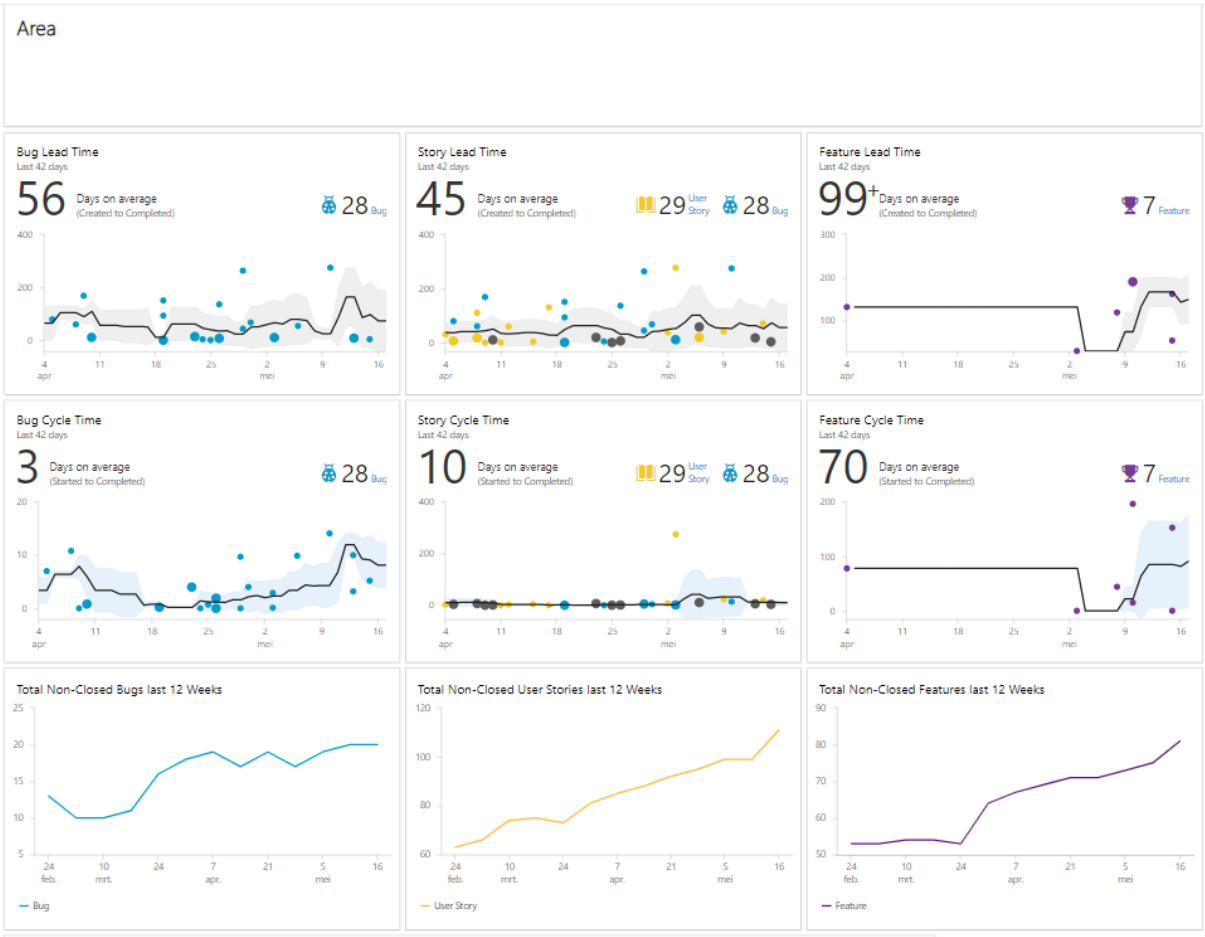
Het tweede onderdeel is een overzicht die de status van de *backlog* gelinkt aan het team moet visualiseren. Om de kwaliteit van de backlog hoog te houden, moeten bugs te allen tijde een ingeschatte hoeveelheid werk hebben en een niveau van ernst. Indien dit niet gebeurt, zijn bugs moeilijker om in te plannen voor sprints. Voor de bugs is er een cirkeldiagram opgesteld die de verhouding van de actieve bugs (Status is niet *Closed*).

Naast het opvolgend van het opvolgen van de kwaliteit van de backlog voor bugs, zijn ook *user story's* in kaart gebracht door bij te houden hoeveel *user story's* geen *acceptance criteria* hebben. Ook het aantal *user story's* zonder een uitleg worden in een kaart aangetoond.

De andere drie kaarten tonen het aan *Epic* topics, *Features* en *User story's* die niet afgerond zijn en gelinkt zijn aan het team. Voor de sprintplanning kan dit gebruikt worden door te kijken naar de output van de query achter de kaarten. Zo kan er werk geselecteerd worden voor volgende sprints.

Onder de kaarten zijn er twee vakken beschikbaar die per feature gelinkt aan het team de *user story's* toont en alle *epic* topics met alle *features* die gelinkt zijn per *epic.*

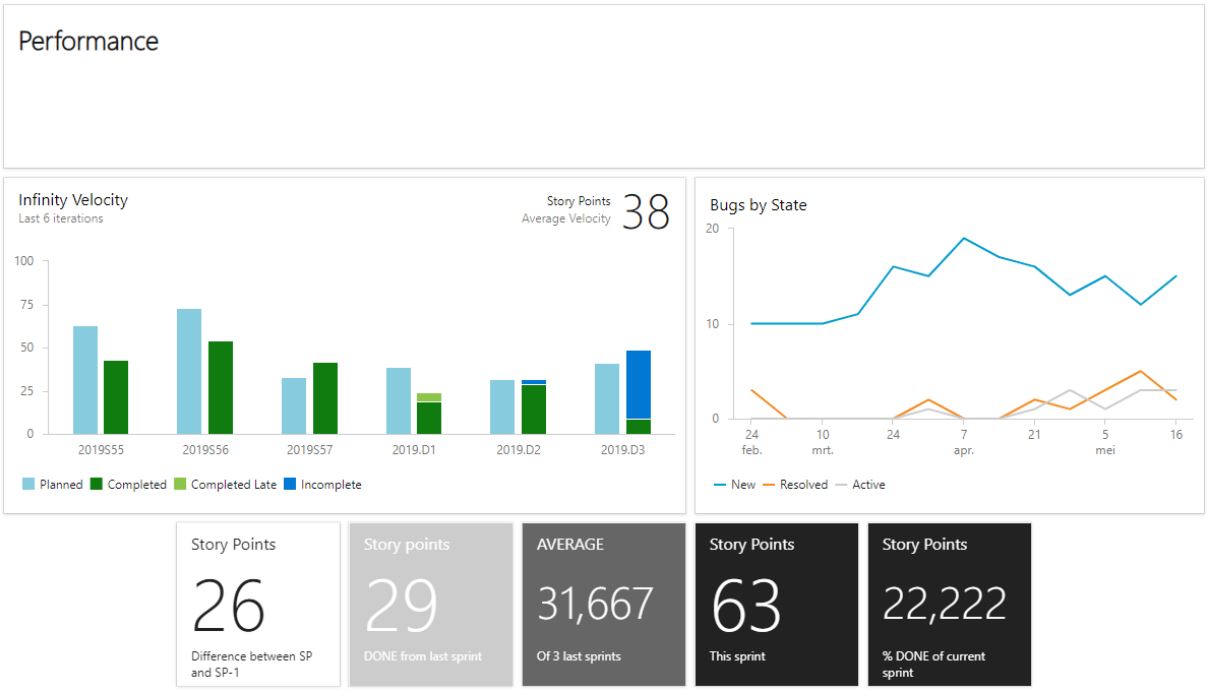
Area overzicht



Figuur 30: Area overzicht van het team dashboard voor één team.

Het area overzicht dient als een algemeen onderdeel om een inzicht te krijgen in de *lead-* en *cycle-*time van de drie belangrijkste *work item* types. Zo wordt dit gedaan voor bugs, *user story's* en *features* die gelinkt zijn aan een team. De visualisatie van de kwaliteitswaarden wordt gedaan door deze te vergelijken voor de laatste 3 sprints. Ook wordt getoond hoeveel van elk type actief zijn doorheen de tijd. Ook dit wordt gedaan voor de laatste drie sprints.

Performance overzicht



Figuur 31: Performance overzicht van het team dashboard voor één team.

Onderaan het teamdashboard is er een overzicht waarbij de huidige sprint vergeleken wordt met vorige sprints op gebied van *velocity*, het verwerken van bugs en voornamelijk het vergelijken van de *story points*. Er worden 5 kaarten gebruikt waarbij de eerste het aantal *story points* die meer of minder opgenomen zijn in de huidige sprint in vergelijking met de vorige sprint. De tweede kaart toont het aantal *story points* van de vorige sprint die afgerond zijn. Een derde kaart toont het gemiddelde afgewerkte *story points* van de laatste drie sprints. De voorlaatste kaart is net als bovenaan het *dashboard* het aantal *story points* die opgenomen zijn in de huidige sprint. Als laatste wordt een percentage getoond van hoeveel *story points* er in de huidige sprint afgerond zijn.

7.2.4 *Interpretatie van het dashboard*

Sprint Overzicht:

Het overzicht van de sprintplanning is een levend onderdeel dat constant verandert in cijfers. Het is de bedoeling dat elk team op elk moment kan zien hoeveel werk er al verricht is, hoeveel werk er actief staat en hoeveel werk er nog gedaan moet worden in de actieve sprint. De kaarten met cijfers moeten helpen bij de analyse van de sprint.

***Backlog* Overzicht:**

In het overzicht van de *backlog* valt op dat er 13% van de bugs die gelinkt zijn aan het team waar het eindwerk in gemaakt is, geen niveau van ernst hebben. Op het moment van de analyse viel op dat de bugs zonder een niveau van ernst dezelfde zijn als die zonder een hoeveelheid werk. Dat wordt aangetoond als er ingekeken wordt in de query's en de ID's vergeleken worden. Ook de kaarten die het aantal *user story's* aantonen zonder *acceptance criteria* en *description* zijn met iets meer dan 10% aanwezig in de *backlog*. In tegenstelling tot de bugs, zijn de ID's van de *story's* verschillend tussen de twee query's.

***Area* Overzicht:**

Bij de grafieken van *lead*- en *cycle time* kan er afgelezen worden dat er een grote *reaction time* is op *items* die in de *backlog* bewaard worden. Voor bugs is er gemiddelde 53 dagen nodig voordat er iemand binnen het team de gemelde bug oplost. Dat terwijl er 3 dagen nodig zijn om de bug op te lossen. Er wordt dus veel verloren tijd bij het starten aan de bug terwijl er aanzienlijk minder tijd nodig is om deze op te lossen.

Hetzelfde besluit kan gesteld worden voor *user story's*. Al is de *reaction time* kleiner in vergelijking met die van bugs, is er nog steeds 35 dagen nodig voordat er aan een *user story* gewerkt wordt. De tijd om een user story af te ronden duurt aanzienlijk langer dan het oplossen van bugs (*cycle time* is 10 dagen).

***Performance* Overzicht:**

Aan de hand van het laatste onderdeel kan een team een inzicht krijgen in de verschillende sprints. Zo kan er vergeleken worden of de huidige sprint productiever was dan de vorige sprints. Daarnaast biedt de *velocity* staafdiagram gedetailleerde informatie over hoeveel werk er gepland was en hoeveel werk er tijdig of te laat afgewerkt is. De trend toont dat er in de laatste 5 sprints er drie sprints waren waar er 20 *story points* niet werden afgewerkt. Daarnaast is er een minimum aantal *story points* die te laat afgewerkt werden. (lichtgroene kleur op het staafdiagram).

7.2.5 *Conclusie proceskwaliteit*

Het opvolgen van de kwaliteit van het proces achter de softwareproductie is een domein dat het best opgevolgd wordt in kleinere groepen. Bij een overzicht per team kan er veel sneller ingegrepen worden wanneer er veranderingen gebeuren die teruggedraaid moeten worden.

Voor Selligent Is het opvallend dat er een heel grote *reaction time* is voor het oplossen van bugs en het afwerken van *user story's*. Daarnaast is er een hoeveelheid *bugs* en *user story's* die niet volledig zijn en nog aangevuld moeten worden om bepaalde eigenschappen een waarde te geven. Om aan deze *items* te werken moet er duidelijk gemaakt worden wat de impact van de *work items* zijn.

7.2.6 *Verdere optimalisatie*

Voor het opvolgen van de proceskwaliteit is er gekozen om voor een individueel team waarden te visualiseren. De *query's* die gebruikt zijn in Azure DevOps zijn opgebouwd om telkens voor het specifieke team de waarden te gebruiken. Om ieder team de kans te geven om het opgebouwde *dashboard* te gebruiken, moet er telkens een nieuwe query gebruikt worden om voor het team de juiste data te tonen.

Om de codekwaliteit van elk team te visualiseren is er echter een link nodig tussen de databron SonarQube en Azure DevOps. Dit is tijdens het eindwerk niet kunnen uitwerken doordat het gebruik van SonarQube niet rechtstreeks in verband staat met wat het team opleverde.

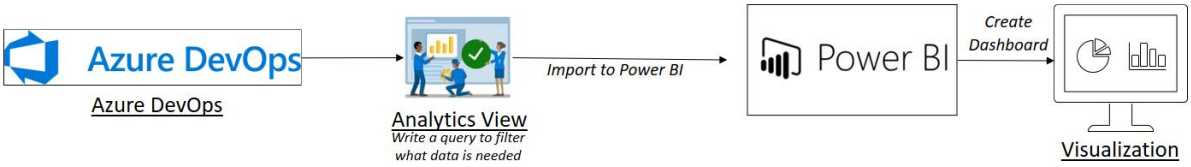
7.3 Productie dashboard

7.3.1 *Inleiding*

Het productiedashboard is een visualisatie die gemaakt is in opdracht van vragende partijen binnen de organisatie die zelf een selectie gemaakt hebben uit de lijst met kwaliteitswaarden. Op basis van de selectie is een apart *dashboard* opgesteld dat in functie staat om verschillende *stakeholders* een eigen inzicht te geven van de staat van de productie.

7.3.2 *Databron diagram*

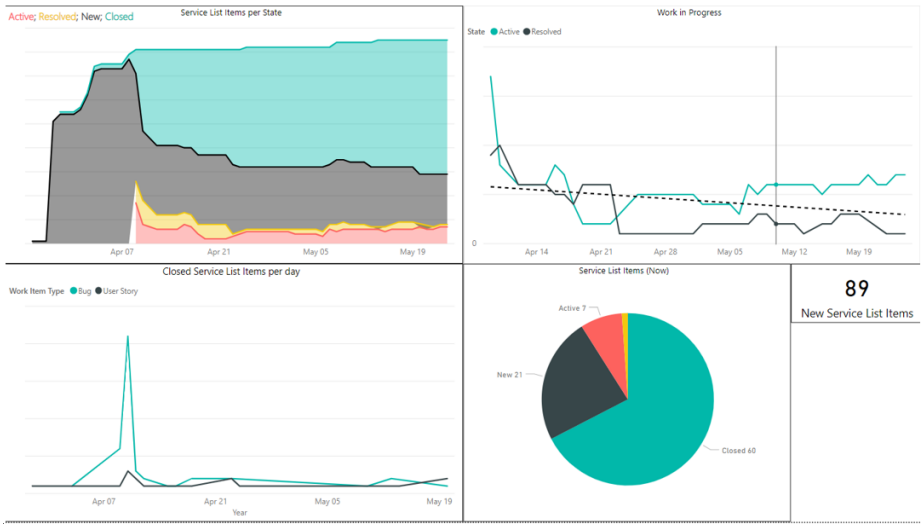
De data die verwerkt en gevisualiseerd wordt is rechtstreeks uit Azure DevOps gehaald met behulp van de *analytics view* functie. Daarna zijn de *datasets* in Power BI opgesteld op basis van de opgestelde *views* van Azure DevOps.



Figuur 32:Databron Productiedashboard.

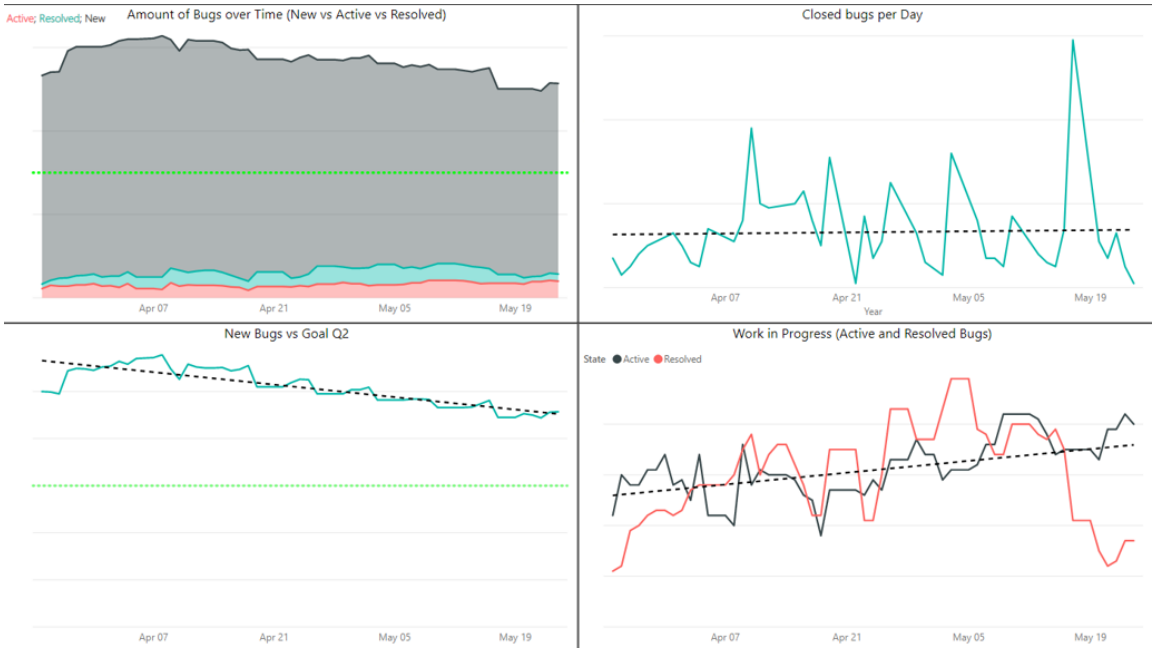
7.3.3 *Uitwerking dashboard*

In opdracht van de *Vice President* binnen de organisatie, is deze visualisatie opgebouwd uit 3 *dashboards* die gebruikt worden om een algemeen overzicht te geven van de productiekwaliteit. Er is een dashboard die de *priority* lijst in kaart brengt. Dit is een lijst met bugs en *user* story's die de hoogste prioriteit op voorhand gekregen hebben om aan te werken. Op het *dashboard* is er een *area chart* links bovenaan geplaatst die moet aantonen hoeveel bugs er per status zijn. Daarnaast is met behulp van een trendlijn het aantal bugs en user story's waar aan gewerkt wordt getoond. Onderaan is er een trendlijn te zien die aantoont hoeveel *work items* er per dag gesloten worden en wordt per staat nog eens getoont hoeveel *work items* er zijn met het exacte aantal items die in de *priority* lijst bewaard zijn. Bepaalde waarden zijn uit de *screenshots* verwijderd om interne data te beschermen.



Figuur 33: Dashboard priority lijst.

Het tweede *dashboard* biedt informatie over de bugs die in de backlog bewaard worden. Dit *dashboard* moet helpen om afgesproken doelen te bereiken voor een bepaalde datum. Net als bij het eerste *dashboard* is er een *area chart* gebruikt voor dezelfde reden. Ook het aantal bugs die opgelost zijn per dag wordt rechts getoond. Het aantal actieve bugs wordt in een lijngrafiek links onderaan getoond met een trendlijn om een zicht te krijgen op welk tempo het bedrijf zijn doel behaald om een specifiek aantal bugs op te lossen. Als laatste wordt ook getoond hoeveel bugs er actief zijn en gecontroleerd moeten worden of ze opgelost zijn per dag.



Figuur 34: Dashboard bugs.

Als laatste is er een *dashboard* gemaakt die de automatisatie van test *cases* met een trendlijn toont. Daarnaast is het maximum aantal manuele test cases gemarkeerd en het doel dat bereikt moet worden. Daarnaast wordt het exacte aantal manuele test *cases* per team in een cirkeldiagram getoond.

Figuur 35: Dashboard manuele test cases.

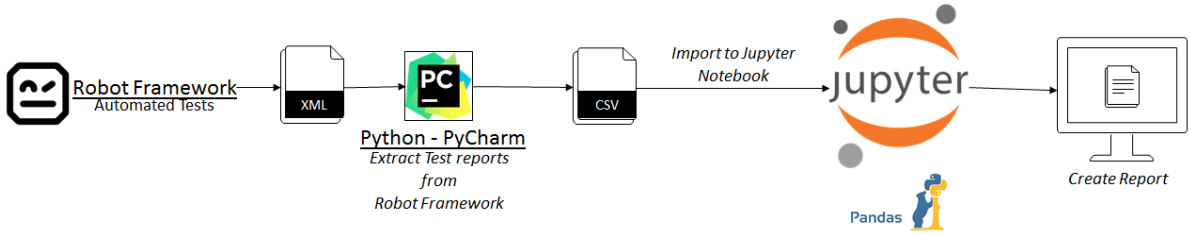
7.4 Rapport automatische testen

7.4.1 *Inleiding*

Om het rapport op te bouwen wordt er niet gewerkt met de visualisatietools Power BI of Azure DevOps. Om QA een snel inzicht te geven in de kwaliteit van de automatische testen, wordt beroep gedaan op de werking van Jupyter Notebook en Pandas.

7.4.2 *Databron Diagram*

De data die nodig is om het rapport op te stellen wordt gehaald van de XML-bestanden die aangemaakt worden door het Robot Framework. Met behulp van een zelfgeschreven script in Python worden de gegevens opgevraagd en omgevormd tot een CSV-bestand. Na het aanmaken van het CSV-bestand wordt deze manueel geüpload naar Jupyter Notebook om daar met de data te werken.



Figuur 36: Databron diagram van Rapport automatische testen.

7.4.3 *Opbouw Rapport*

Het rapport wordt opgesteld om een onderscheid te hebben in testen die instabiel zijn en welke robuust zijn. Binnen de organisatie wordt er gesproken van *Flaky* testen wanneer deze instabiel zijn. Per definitie wordt een test *flaky* genoemd wanneer die een bepaald aantal keren faalt in verhouding met het totaal aantal keren dat deze doorlopen wordt. Stabiele testen worden *Stable* genoemd.

Met het script wordt data opgevraagd van telkens de eerste test *run* van de afgelopen 10 dagen. Er wordt in het script een lijst bijgehouden van datums die overgeslagen moeten worden indien deze irrelevant zijn om in het rapport te verwerken.

Nadat de data opgehaald is met het script, wordt met Jupyter Notebook en Pandas de verwerking gedaan. Met de ruwe data die beschikbaar gemaakt wordt, wordt een overzicht gemaakt van alle test *cases* gelinkt aan hun *suite*. Daarbij wordt voor elke test case getoond hoeveel keer deze doorlopen is en hoeveel keer deze slaagt of faalt. Op basis van het aantal keren dat een test case faalt kan er geoordeeld worden of deze *flaky* of *stable* is. Alvorens dit beslist is via het rapport is er al een onderscheid gemaakt door *flaky* testen in het veld *Tags* een tag 'FLAKY' te geven. Indien de tag niet aanwezig is, dan is de test case *stable*.

01 Api	s1-s1	Request_POST_Entrypoints_Details	s1-s1-s3-t3	PASS	['API', 'FLAKY', 'JOURNEY', 'NEW', 'REGRESSION...

Figuur 37: Voorbeeld Test Case.

Als controle op de voorafgaande beslissingen om test *cases* als *flaky* te bestempelen of niet, moet via dit rapport een inzicht gegeven worden om die test *cases* en aantonen of die beslissingen kloppen. Als eindresultaat van het rapport zijn er vier onderdelen waarvan het aantal test *cases* beschreven wordt en een lijst van deze test *cases*. De vier onderdelen worden op dezelfde manier weergegeven en wordt dus met één enkel voorbeeld in dit eindwerk besproken. De onderdelen zijn: *flaky* testen die *flaky* zijn (*True Flakies*), *flaky* testen die eigenlijk *stable* horen te zijn (*False Flakies*), *stable* testen die eigenlijk *flaky* horen te zijn (*False Stables*) en *stable* testen die *stable* zijn (*True Stables*). De sortering van de test cases per categorie gebeurt door te stellen of het aantal geslaagde testen 80% of meer van het totaal aantal *cases* zijn. Indien dit klopt wordt in de tabel een kolom voorzien die een *boolean* waarde bijhoudt van deze vergelijking.

Voor het onderstaande voorbeeld is dus de vergelijking gemaakt of er een tag 'FLAKY' aanwezig is en het aantal geslaagde test *cases* lager dan 80% is.

```
sum((df_final['Flaky Test']==True) & (df_final.Stable==False))
```

Figuur 38: Bepalen van 'true flaky' test cases.

Voor elke categorie wordt in het begin het aantal van die test *cases* getoond. Als volgende onderdeel wordt er een tabel uitgeprint die alle test *cases* in een lijst weergeeft. Hierbij wordt voor elke test *case* de *suite* getoond waaraan de *case* gelinkt is, het aantal geslaagde en gefaalde testen, de naam en id van de test case, of de test case al dan niet *flaky* is met behulp van een boolean waarden, het slaagpercentage van de test *case* en de kolom die bepaalt of het slaagpercentage groter is dan 80% om als *stable* gecategoriseerd te worden.

True Flakies

'Flaky tests that are flaky: 12'

	File Name	Suite Name	Suite id	Test Name	Test ID	Status	Passes	Fails	Total	Flaky Test	Stable Test	Passed_Percentage	Stable
8			s1-s4		s1-s4-s1-t1	FAIL	4.0	6.0	10.0	True	False	40.0	False
10			s1-s4		s1-s4-s5-t1	FAIL	0.0	10.0	10.0	True	False	0.0	False
11			s1-s4		s1-s4-s4-t3	FAIL	0.0	10.0	10.0	True	False	0.0	False
12			s1-s4		s1-s4-s4-t2	FAIL	0.0	10.0	10.0	True	False	0.0	False
13			s1-s4		s1-s4-s4-t1	FAIL	0.0	10.0	10.0	True	False	0.0	False
14			s1-s4		s1-s4-s4-t4	PASS	7.0	3.0	10.0	True	False	70.0	False
18			s1-s4		s1-s4-s3-t4	PASS	5.0	5.0	10.0	True	False	50.0	False
21			s1-s4		s1-s4-s3-t5	PASS	5.0	5.0	10.0	True	False	50.0	False
23			s1-s4		s1-s4-s5-t2	FAIL	0.0	10.0	10.0	True	False	0.0	False
31			s1-s5		s1-s5-s4-t2	PASS	6.0	4.0	10.0	True	False	60.0	False
33			s1-s5		s1-s5-s6-t1	PASS	6.0	4.0	10.0	True	False	60.0	False
39			s1-s4		s1-s4-s7-t1	FAIL	0.0	10.0	10.0	True	False	0.0	False

Figuur 39: Output in Jupyter Notebook van 'True Flaky' test cases.

7.5 Sales Dashboard

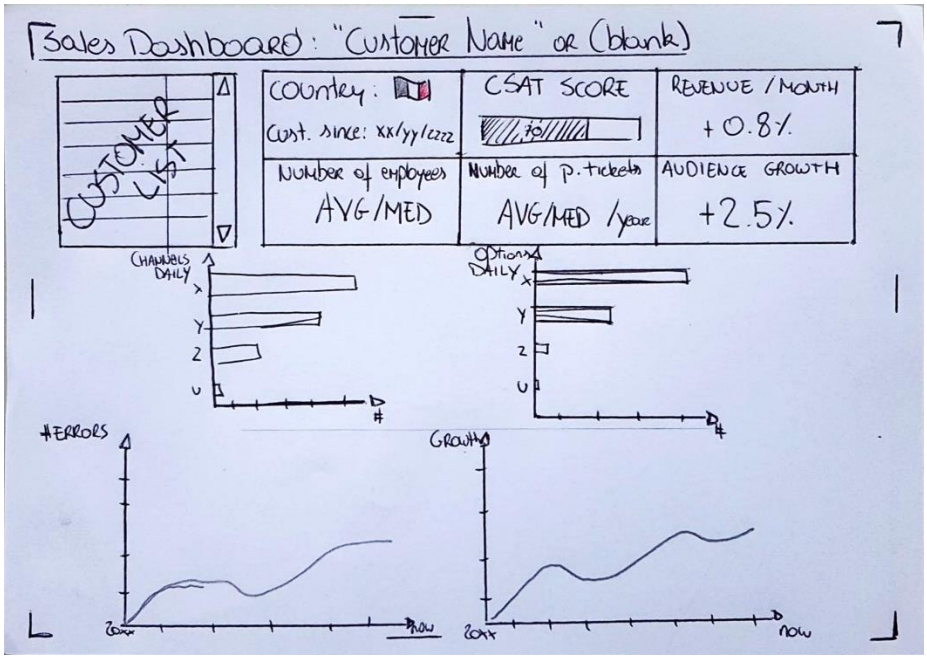
7.5.1 *Inleiding*

Voor *Sales* is het belangrijk om een zicht te krijgen op *trends*. Dat zowel van alle klanten in het algemeen als specifiek per klant. Omdat het bouwen van een *dashboard* voor *Sales* buiten de *scope* van het eindwerk valt, is er enkel een *mockup* getekend die een beeld moet geven van hoe het *dashboard* er uit kan zien. De *mockup* is opgebouwd met als mentaliteit om deze uit te werken in Power BI.

7.5.2 *Mockup*

Links bovenaan moet er een tabel aanwezig zijn waarin alle klanten opgesomd zijn met enkel de naam. Indien er geen klant geselecteerd wordt in de tabel, is het *dashboard* een beeld van dezelfde waarden maar dan voor alle klanten.

Indien er een klant geselecteerd wordt, wordt voor die klant bovenaan het land weergegeven en hoelang een klant al met software van Selligent werkt. Voor *Sales* is het belangrijk dat wanneer klanten uniek zijn in hun land, deze goed behandelt worden om de groei in dat land te bevorderen. Daarnaast is het belangrijk om een idee te krijgen van hoe groot een organisatie is. Er wordt dan ook het aantal werknemers in een ander kader getoond. De CSAT-Score is een algemeen cijfer dat gegeven wordt aan klanten om een beeld te scheppen van hoe hun ervaringen met Selligent zijn. Dat cijfer wordt door meerdere factoren bepaalt en wordt niet verder uitgewerkt in dit eindwerk. Voor *Engineering* en *Sales* is het nuttig om te weten hoeveel *pink tickets* een klant gemiddeld doorgeeft per tijdsperiode (weken, maanden, jaren, …). Op basis hiervan kan er geoordeeld worden hoe geëngageerd een klant is om te helpen bij het verbeteren van de software. Als laatste zijn er vakken beschikbaar waar de winst van het bedrijf ingeschat wordt en wat de groei van de klant zijn publiek is, om te kunnen begrijpen wat de impact van Selligent is op die klant.



Figuur 40: Mockup Salesdashboard. (Dummy Data)

II. Onderzoekstopic

1 Probleemstelling

Tijdens het productieproces kunnen er onverwachte obstakels voorkomen in de applicatie die een probleem kunnen vormen voor de klanten. Het is van belang om deze obstakels nauw op te volgen en zo snel mogelijk op te lossen. Door het in kaart brengen van deze obstakels kan de kwaliteit van zowel product als het proces erachter geoptimaliseerd worden.

Welke waarden binnen het softwareontwikkelingsproces zijn dan van belang om de kwaliteit van zowel het product als het proces op te volgen? En welke waarden binnen het softwareontwikkelingsproces zijn van belang om de kwaliteit van zowel product als proces op te volgen?

Momenteel zijn er meerdere platformen waarop *Software-engineers* werken bij Selligent. Dat is erg nadelig om op een efficiënte manier de kwaliteit te visualiseren van het proces als het product. Het primaire platform waarop gewerkt wordt is Azure DevOps. Het grootste deel van de nodige data wordt dus ook teruggevonden op dit platform. Echter zijn er nog andere platformen waar productieprocessen aanwezig zijn en waar er nog geen link is met de data op Azure DevOps. Er wordt gezocht naar manieren om uit alle gebruikte platformen de data te verkrijgen die nodig zijn om een volledig beeld te krijgen van de kwaliteit.

Voor de *Software-engineers* is het van belang om op een efficiënte manier een visueel beeld te krijgen van de staat van het product waaraan gewerkt wordt. Momenteel kan elk team op een eigen manier een *dashboard* opbouwen van wat er enkel beschikbaar is op Azure DevOps. Andere data die gevisualiseerd moet worden per team, moet dus op een alternatieve manier bereikt worden.

Door het opbouwen van een uniform *dashboard* rond de belangrijkste kwaliteitswaarden heeft iedereen een zelfde beeld van de staat van het product. Hierdoor kunnen er betere beslissingen gemaakt worden en kan er een efficiëntere werkverdeling gebeuren om proactief te handelen op onverwachte obstakels.

Welke verschillen zijn er tussen visualisatietools die de potentie hebben om gebruikt te worden voor het *dashboard*? En hoe worden anomalieën blootgelegd via het de visualisaties*?*

Om een antwoord te krijgen op de vragen wordt met behulp van een bronnenonderzoek een lijst opgesteld van relevante waarden die een link hebben met softwarekwaliteit. Na het opstellen van deze lijst wordt een selectie gemaakt van de belangrijkste waarden die er op een *dashboard* aanwezig moeten zijn.

Nadat bekend is welke waarden er gevisualiseerd moeten worden, wordt gezocht naar de geschikte tool die op een eenvoudige manier de kwaliteit van het product als proces in kaart kan brengen. Door te werken met verschillende tools worden er verschillen gezocht tussen de tools en wordt één tool geselecteerd waar er mee gewerkt wordt.

Met de geselecteerde tool wordt een *dashboard* opgesteld en wordt beschreven welke anomalieën er aangetoond kunnen worden met behulp van de visualisatie.

2 Onderzoeksmethode

Het onderzoek start met een bronnenonderzoek waarbij er met behulp van beschikbare informatie een lijst opgesteld wordt die alle relevante waarden beschrijft die gelinkt zijn met softwarekwaliteit. Die waarden moeten helpen bij het opbouwen van een dashboard die de kwaliteit in één oogopslag beschikbaar moet stellen. Ook wordt het productieproces binnen de organisatie geschematiseerd om alle opgenoemde kwaliteitswaarden de lokaliseren binnen de productie. Alle kwaliteitswaarden zullen op een uniforme manier beschreven worden om de lijst zo helder mogelijk te beschrijven.

Na het identificeren van alle relevante kwaliteitswaarden, moet er duidelijk gemaakt worden waar alle nodige brondata opgeslagen is. Er moet duidelijk gesteld worden waar de brondata is en hoe deze opgevraagd kunnen worden.

Na het opstellen van de lijst met kwaliteitswaarden en te weten waar de brondata bewaard wordt, wordt een selectie gemaakt uit de waarden die de grootste impact kunnen geven bij het visualiseren en optimaliseren van de softwarekwaliteit. Op basis van deze selectie kan er een visualisatie gebeuren van de kwaliteitswaarden met behulp van een *dashboard* dat opgesteld wordt met een gekozen tool.

Er worden drie tools vergeleken die de potentie hebben om te gebruiken voor het opbouwen van een *dashboard*. Als eerste wordt gekeken naar het huidig gebruikte Azure DevOps. Een tweede tool die onderzocht wordt is Power BI. Als laatste wordt een alternatieve manier onderzocht door met Python te werken. Zo wordt met behulp van een combinatie van een *library* Pandas en *tool* Plot.ly gebruik gemaakt.

De drie opties worden vergeleken met elkaar op basis van hun functionaliteiten en gebruiksvriendelijkheid. Als besluit wordt een selectie gemaakt uit één van de tools waar er mee verder gewerkt wordt.

Na de selectie van de tool wordt een *dashboard* opgebouwd die gebruik maakt van de lijst met kwaliteitswaarden door een selectie te maken van de belangrijkste waarden om te visualiseren. Als laatste wordt uitgelegd welke anomalieën er in kaart gebracht worden met behulp van het *dashboard*.

3 Kwaliteitswaarden

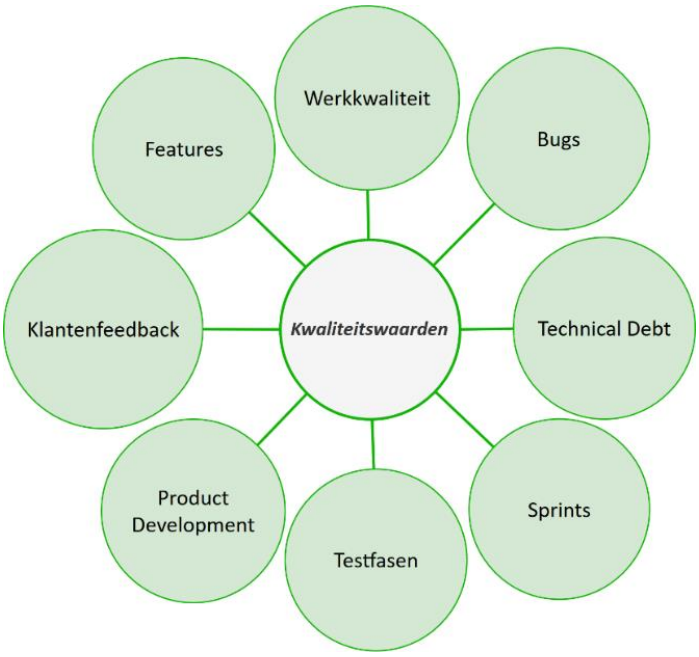
3.1 Literatuurstudie

Voor het verzamelen van de kwaliteitswaarden die beschreven zijn in het onderzoekstopic, zijn verschillende bronnen van informatie gebruikt die in een lijst opgesomd staan. De meerderheid aan kwaliteitswaarden zijn uit het boek *The Science of Lean Software and DevOps: Accelerate (Bron 7)* gehaald.

3.2 Inleiding

Als voorbereiding op het visualiseren van kwaliteitswaarden, moet er onderzocht worden welke waarden van belang zijn om de kwaliteit van het proces als het product te kunnen evalueren. In dit hoofdstuk is er een lijst opgebouwd die onderverdeeld is in acht thema's. De thema's worden gebruikt als link tussen de waarden die gevisualiseerd op het dashboard en de waarden die in de lijst opgesomd zijn. De verdeling van de kwaliteitswaarden onder de verschillende thema's is beslist op basis van het groeperen en linken van de verschillende waarden. Met behulp van het bronnenonderzoek zijn de verschillende waarden in een lijst opgesomd.

3.3 Verzamelen van waarden



Figuur 41: Onderverdeling kwaliteitswaarden.

Elke waarde wordt beschreven in dezelfde structuur om het overzicht over alle waarden uniform te maken. Volgende elementen zijn besproken:

- Definitie: Een beknopte uitleg waarvoor de titel van de waarde staat en waarom de waarde relevant is om de kwaliteit op te meten.
- Berekening: De beschrijving die aantoont hoe de waarde opgemeten kan worden. Daarnaast wordt ook uitgelegd waar de brondata van de waarde gelokaliseerd is.
- Relevantie: De relevantie toont aan of er andere kwaliteitswaarden gelinkt kunnen worden aan de beschreven waarde om een resultaat beter in kaart te brengen. De relevantie wordt enkel in kaart gebracht als er waarden relevant zijn.

- Voorbeeld van visualisatie: Een voorbeeld van hoe de visualisatie van de kwaliteitswaarde kan zijn. Een visualisatievoorbeeld wordt enkel getoond indien dit mogelijk was te bereiken tijdens het onderzoek. Bij visualisatievoorbeelden zijn bepaalde waarden verwijderd uit de voorbeelden om interne data te beschermen.

3.4 Product Development

Onder *Product Development* zijn waarden besproken die helpen bij het opvolgen van de kwaliteit van het ontwikkelingsproces van de verschillende applicatieonderdelen.

3.4.1 *Levensduur van de workbranch: [7]*

Definitie: De levensduur van een actieve *workbranch* moet als doel hebben om maximaal één werkdag actief te zijn. Het doel van de levensduur is om te streven naar c*ontinuous delivery.* Door te werken naar *continuous delivery* wordt ook rechtstreeks gewerkt aan de werkkwaliteit binnen de organisatie.

Berekening: Als aanbeveling voor de organisatie moet er een overzicht komen over de levensduur van *workbranches*. [7] Een *lead time* van elke *branch* moet helpen bij het creëren van een overzicht voor elke *workbranch*. Elke *workbranch* wordt bijgehouden in een *repository*. De tijd van creatie tot het afronden van deze *branch* zijn gegeven velden die automatisch aangevuld worden in Azure DevOps. Door het aantal *workbranchen* per sprint te vergelijken, kan er een lijst opgesteld worden met welke branches langer dan één dag actief waren.

Relevantie: Indien blijkt dat er *workbranchen* langer actief zijn dan één werkdag, is het relevant om te kijken naar welke *work items* gelinkt zijn aan de *branch*. Bepaalde *work items* zullen een te grote werkballast hebben om af te werken binnen één werkdag.

3.4.2 *Actieve branches per team [7]*

Definitie: Om de werkkwaliteit optimaal te houden en de werkballast niet te moeten overdragen aan toekomstige sprints, is het aangeraden om een maximum van drie actieve branches te hebben per team. Wanneer er gestreefd wordt naar dit doel, wordt automatisch ook gewerkt aan *continuous delivery.*

Berekening: Als aanbeveling voor de organisatie is het geschikt om voor elk team een *tag* met de teamnaam te plaatsen bij de *workbranch*. Dat helpt om het aantal *branches* te tellen per team. Het is belangrijk dat teams niet met elkaar vergeleken worden om geen competitieve sfeer te creëren binnen de organisatie. Het is voor elk team van individueel belang om hun eigen *workbranchen* te evalueren.

Relevantie: Net als bij 3.3.1 (Levensduur van de workbranch) biedt het een beter inzicht om de actieve *branches* te linken aan de *work items* waarvoor gewerkt worden. Zo kan er een inzicht per team gecreëerd worden over de werkkwaliteit.

3.4.3 *Status van pull request [7]*

Definitie: Voor de *pull request* is het belangrijk om bij te houden of deze als status '*Active*' heeft. Het aantal die deze waarde hebben tellen mee als onderdeel van het werk dat nog uitgevoerd moeten worden. Wanneer er een overzicht is van de actieve *pull requests*, kan er bepaald worden hoe de kwaliteit van werkuitvoering is tijdens de ontwikkeling van de applicatie.

Berekening: Binnen het platform Azure DevOps worden actieve *pull requests* weergegeven in een lijst. Alle actieve *pull requests* kunnen opgeteld worden. Afhankelijk van het team dat een zicht hierover wilt hebben, kan er per team gekeken worden naar het aantal.

3.4.4 *Tijd gespendeerd aan productvoorbereiding [7]*

Definitie: De tijd die gespendeerd wordt aan het analyseren van het pro, het opstellen van *requirements* en andere voorbereidingen. Door deze waarde op te meten, kan er een inzicht verkregen worden in de werkkwaliteit binnen de organisatie. De balans tussen productvoorbereiding en *product development* moet juist zijn zodat er een constante progressie is

Berekening: Elke *work item* krijgt een hoeveelheid aan *story points*. Voor elke sprint moet er nagegaan worden welke *work items* gelinkt zijn aan productvoorbereiding. Zo kan het onderscheid met product development gemaakt worden.

Relevantie: Om deze waarde duidelijk in kaart te brengen, is het van belang om het aantal *work items* gelinkt aan product *development* te betrekken bij de visualisatie.

3.4.5 *Tijd gespendeerd aan product development [14][7]*

Definitie: De tijd die gespendeerd wordt aan het schrijven van code, testen van *features* en het herstellen van belangrijke functionaliteiten. Die waarde kan een inzicht geven in de werkkwaliteit binnen de organisatie. Indien er een slechte balans is tussen productvoorbereiding en *product development*, moet er geëvalueerd worden wat er moet gebeuren om de balans terug goed te krijgen.

Berekening: Net zoals bij 3.3.4 is de berekening via het optellen van de gelinkte *story points.*

Relevantie: Het aantal *story points* die gelinkt zijn aan productvoorbereiding helpen om de vergelijking van alle *work* items te maken voor de sprint. De twee waarden samen (3.3.4 en 3.3.5) bieden een inzicht om meerdere sprints met elkaar te vergelijken. Uit de vergelijking kan afgeleid worden of er vooruit- of achteruitgang geboekt is in de verhouding tussen product *development* en productvoorbereiding.

3.4.6 *Deployment Frequentie [14][7]*

Definitie: *Deployment* frequentie voor een applicatie is de waarde die aantoont hoe frequent er veranderingen gebeuren aan de applicatie. Om te streven naar *Continuous Delivery* is het belangrijk om deze zo hoog mogelijk te hebben per uur, dag, week, … afhankelijk van wat het doel is voor de teams en organisatie.

Berekening: Wanneer er een nieuwe verandering gebeurt aan de *repository*, moet het aantal dagen berekend worden tussen de datum van de vorige en nieuwe verandering. Nadat dit aantal gekend is, wordt berekend wat het gemiddelde is over een bepaalde periode. Als visualisatie is het geschikt om een lijngrafiek te gebruiken die aantoont hoe frequent er veranderingen aan een *repository* gebeurt.

Relevantie: Wanneer een *deployment* faalt of er is een aanzienlijk verschil in de frequentie, is het van belang om de *deployment* resultaten te betrekken bij de visualisatie om de oorzaak te vinden.

3.4.7 *Deployment Fail Ratio [7]*

Definitie: De vergelijking tussen het aantal geslaagde en gefaalde *deployments* bieden een inzicht over de kwaliteit van de applicatie doorheen de tijd. Die waarde biedt hulp bij het reageren op slechte resultaten door tijdig op te merken wanneer het fout loopt. Op basis van deze visualisatie kan er bij andere waarden nagekeken worden wat de oorzaak is van meerdere gefaalde *deployments* achter elkaar.

Berekening: Wanneer er een *merge* gebeurt met de nieuwe code, wordt dit automatisch bijgehouden in Azure DevOps. Het resultaat hiervan wordt bijgehouden in een vergelijking tussen het aantal geslaagde en gefaalde *deployments.* Het slaagpercentage van *deployments* wordt bij een visualisatie in een tekstvak weergegeven van de actieve sprint.

Relevantie: Indien er gefaalde *deployments* zijn, is het van belang om te kijken naar de gevonden *bugs* en de kwaliteitswaarden die beschreven zijn onder het onderdeel *bugs.*

3.4.8 *Gedupliceerde code [3]*

Definitie: Gedupliceerde code is een aantal lijnen code die op verschillende plekken in een applicatie opnieuw geschreven zijn. [22] Een applicatie werkt efficiënter wanneer dit zoveel mogelijk gemeden wordt. Daarnaast maakt het de code voor *Software-engineers* overzichtelijker om eventuele wijzigingen door te voeren.

Berekening: De output van deze waarde wordt bepaald in rapporten van SonarQube. Als aanbeveling is het geschikt om een API te gebruiken die deze waarde kan uitlezen en kan verwerken in een leesbaar bestand. Eens de output van deze waarde beschikbaar is, wordt deze getoond in een tekstvak voor de actieve sprint of in een lijngrafiek per sprint.

Relevantie: Deze waarde gecombineerd met 3.3.9 (Code *smells*) biedt een extra inzicht in de kwaliteit van de geschreven code per *build* of *release*.

3.4.9 *Code Smells [3]*

Definitie: *Code Smells* zijn zwakke punten in het ontwerp van de applicatie die op lange termijn problemen kunnen vormen. De kwaliteit van de applicatie wordt mede bepaalt door het aantal *code smells* die aanwezig zijn in de applicatie.

Berekening: Deze waarde is een berekend veld in een rapport van SonarQube. Een aanbeveling is dat er met behulp van een API gewerkt wordt. Op deze manier kunnen de waarden opgevraagd worden en in een leesbaar formaat bijgehouden. Als de output beschikbaar is, wordt deze voor actieve sprints in een tekstvak weergegeven of wordt deze per sprint in een lijngrafiek gevisualiseerd.

Relevantie: Deze waarde in combinatie met 3.3.8 (Gedupliceerde code) geeft een extra inzicht in de kwaliteit van de geschreven code per *build* of *release*.

3.5 Bugs

Bugs zijn fouten in de applicatie die zo vroeg mogelijk opgespoord moeten worden om de impact zo minimaal mogelijk te houden. Om bugs op te sporen zijn er in het productieproces meerdere perioden voorzien om de ontwikkelde code te testen op bugs. Dat thema van de kwaliteitswaarden is gericht op het inschatten van de aanwezig bugs in de applicatie en de manieren om deze op te sporen. In bijlage A kan er een menu gezien worden van hoe een *work item* opgesteld is in de *backlog* van Azure DevOps.

3.5.1 *Bug Density [1]*

Definitie: Bug Density staat voor de hoeveelheid bugs die aanwezig zijn per applicatieonderdeel. Hoe meer bugs er in een onderdeel zijn, hoe hoger de *density.*

Berekening: In Azure DevOps heeft een bug een veldtype *Product Area*. Dat veld toont aan in welk onderdeel van de software het zich bevindt. De bugs worden opgeteld en gesorteerd per *Product Area* om een inzicht te krijgen in de *bug density.*

Het visualiseren van deze waarde gebeurt met behulp van een cirkeldiagram die de verhoudingen per *product area* aantoont.

Relevantie: In combinatie met de ernst en werkballast van de bugs, helpt deze waarde bij het bepalen van de applicatiekwaliteit per onderdeel.

3.5.2 *Lead Time van Bugs [7][11]*

Definitie: De tijd die gespendeerd wordt aan het oplossen van bugs. *Lead Time* is de tijd die wordt bepaald vanaf de start dat een bug in de *backlog* geplaatst wordt totdat deze opgelost is. Die waarde is een factor binnen een organisatie waarmee het tempo geëvalueerd kan worden om bugs op te lossen en gemeld zijn in de *backlog*.

Berekening: *Lead time* wordt berekend als het aantal dagen tussen de datums wanneer een bug aangemaakt is en wanneer deze gesloten wordt. De datums hebben eigen velden binnen Azure DevOps genaamd *Created Date* en *Closed Date*.

Wanneer deze waarde gevisualiseerd wordt, is het interessant om de minimum, maximum en gemiddelde tijd aan te geven. Die waarden zijn visueel zichtbaar op een lijngrafiek wanneer de lead time per sprint gevisualiseerd wordt.

Relevantie: Naast de *lead time* is *cycle time* ook relevant om te combineren in de lijngrafiek. Dat biedt een extra inzicht in de kwaliteit van het verwerken van bugs.

Voorbeeld:

Figuur 42: Lead time van bugs in 2017, 2018, 2019.

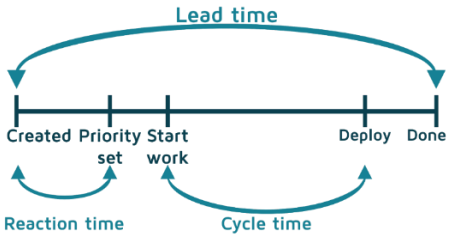
3.5.3 *Cycle Time van Bugs [7][11]*

Definitie: In vergelijking met *lead Time,* is c*ycle time* de tijd die gelinkt is aan het effectieve werk dat verricht wordt om een bug op te lossen.

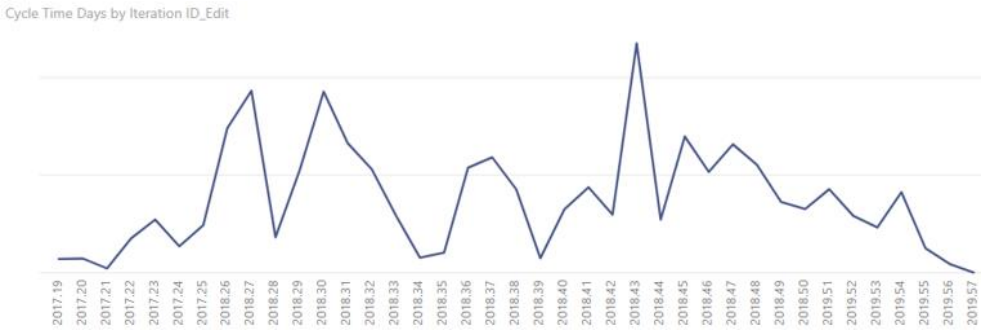
Berekening: Het berekenen van *cycle time* gebeurt aan de hand van de data wanneer een bug zijn status verandert van *New* naar *Active* en van *Active* naar *Closed.* De dagen tussen deze twee data geldt als de *cycle time*. Net als *lead time* is het interessant om de minimum, maximum en gemiddelde tijd te kunnen visualiseren (per sprint, per maand, …) om veranderingen in kaart te kunnen brengen.

Relevantie: In combinatie met *lead time* bieden deze waarden een extra inzicht in de kwaliteit van het oplossen van bugs. Daarnaast kan de *reaction time* berekent worden met behulp van *lead* en *cycle time*. *Reaction time* is het aantal dagen die er zijn tussen het aanmaken van de bug en het starten met het oplossen er van.

Voorbeeld:



Figuur 43: Vergelijking lead, reaction en cycle time. [15]



Figuur 44: Cycle time van bugs in 2017, 2018 en 2019.

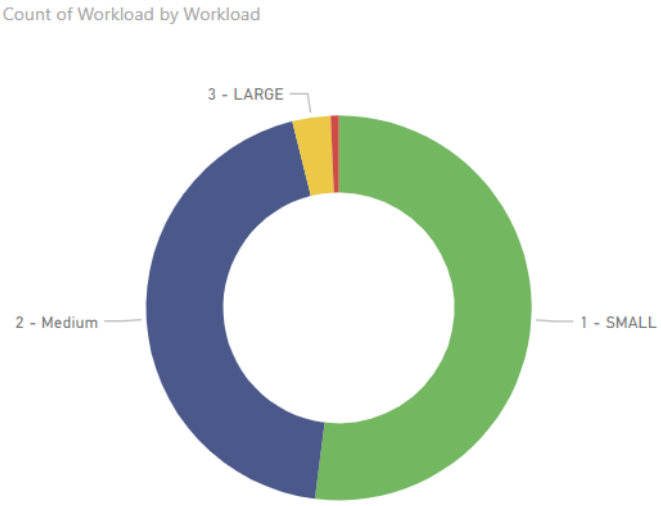
3.5.4 *Werkballast van Bugs*

Definitie: De werkballast per bug toont aan hoeveel werk er wordt ingeschat om deze op te lossen. Die waarde wordt meegegeven aan een bug bij het aanmaken er van. De kwaliteit van de applicatie kan verbeterd worden door een inzicht te hebben in de hoeveelheid werk gelinkt aan de bugs.

Berekening: Elke bug krijgt bij het aanmaken een vaste waarde van werkballast van één tot en met vier (één weinig, vier is heel veel). De vergelijking gebeurt per categorie om een beeld te krijgen van alle bugs gecategoriseerd per werkballast.

Relevantie: De werkballast van bugs gecombineerd met de werkballast van andere *work item* types is een hulp voor een goede sprintplanning.

Voorbeeld:



Figuur 45: Werkballast van bugs aanwezig in de backlog.

3.5.5 *Staat van Bugs [11]*

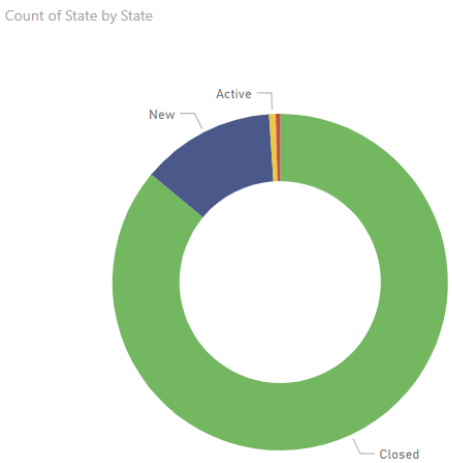
Definitie: De staat waarin een bug zich bevindt, wordt aangegeven door het veld *State*. Dat veld is een standaardwaarde die ingevuld wordt in Azure DevOps. Op basis van de statuscategorieën wordt een inzicht gecreëerd over het aantal bugs die reeds zijn opgelost. Dat kan de werkkwaliteit bepalen per tijdsduur (per sprint, maand, …). De staat wordt in de *backlog* bijgehouden als *New, Active, Closed* of *Resolved.*

Berekening: Om deze waarde te visualiseren moeten alle bugs per status gecategoriseerd worden. Afhankelijk van wat de gebruiker wilt zien, kan er per sprint, release, maand, enz. doorheen de tijd gekeken worden naar de vooruitgang.

De visualisatie gebeurt voor de actieve sprint en de verhoudingen tussen de *work items* per staat. Een cirkeldiagram is het meest geschikt hiervoor.

Relevantie: De ernst en werkballast van bugs helpen bij het in kaart brengen van de werkkwaliteit om bugs op te lossen. Daarnaast is er een mogelijkheid om per productonderdeel te kijken naar de statussen van de bugs.

Voorbeeld:



Figuur 46: Statussen van bugs aanwezig in de backlog.

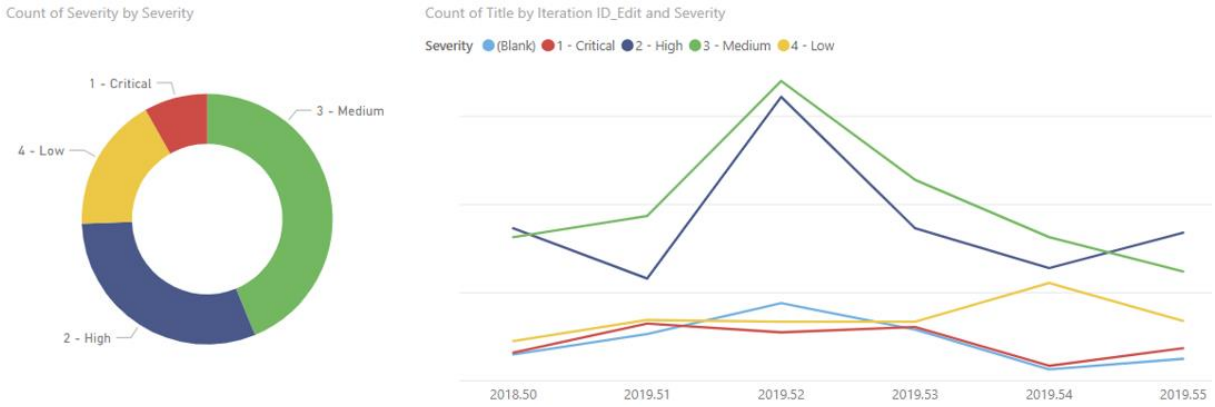
3.5.6 *Ernst van Bugs*

Definitie: De ernst van een bug is een gegeven veld in de eigenschappen van een *work item*. De veldwaarde staat voor het risico dat de organisatie oploopt om deze niet op te lossen. De ernst krijgt een waarde tussen één en vier (één is laag, vier is kritiek). Wanneer deze waarde in kaart gebracht wordt, wordt bepaald in welke toestand de applicatie is op gebied van risico's die gelinkt zijn aan bugs.

Berekening: Om de visualisatie mogelijk te maken wordt een cirkeldiagram gebruikt om de huidige toestand van de applicatie weer te geven. Het aantal bugs wordt opgeteld en moet dan per categorie gesorteerd worden. Daarnaast kan een lijngrafiek aantonen wat de staat van de applicatie is per tijdsperiode (sprint, weken, maanden, etc.).

Relevantie: De ernst van bugs kan ook relevant zijn om een inzicht in te krijgen per product onderdeel. Daarnaast kan een overzicht doorheen de tijd ook interessant zijn om een zicht te krijgen op de vooruitgang om bugs op te lossen per categorie.

Voorbeeld:



Figuur 47: Ernst van bugs aanwezig in backlog (links) en ernst vergeleken per sprint (rechts).

3.5.7 *Escape Rate van Bugs [1][12]*

Definitie: De *escape rate* van bugs toont aan hoeveel gekende bugs terugkomen in de applicatie ondanks deze tijdens vorige productiefasen opgelost zijn. Het aantal wordt weergegeven als een exact aantal, of een percentage in verhouding met alle bugs. De gekende bugs worden gevonden tijdens test fasen. Omdat *regression* testen het doel heeft om dit soort problemen op te sporen, vindt deze waarde zijn nut voornamelijk tijdens deze testfase.

Berekening: Bugs hebben net als alle andere *work items* een ID. Indien er een gekende bug terugkomt, wordt dit ID hergebruikt. Daarbij wordt de staat terug herzien en verandert naar *New*. Als aanbeveling wordt het aantal keren dat een bug terug naar *New* verandert bijgehouden. Per sprint of release wordt het aantal bugs die van *Closed* naar *New* verandert zijn opgeteld. De visualisatie gebeurt door het exacte aantal of het percentage per sprint te tonen in een tekstvak of door deze per sprint in een lijngrafiek te vergelijken.

3.5.8 *Bugs gevonden tijdens Unit Testen [7][13]*

Definitie: Tijdens *unit* testen ligt de focus op het opsporen van bugs en andere fouten in de applicatie. Door op te volgen hoeveel bugs er gevonden worden, is het mogelijk een overzicht te creëren van de kwaliteit van *unit testen*.

Berekening: Om de impact van de unit testen te begrijpen, moet het aantal bugs vlak voor de unit testfase bijgehouden worden. Na een *run* van de *unit* testen moet het aantal opnieuw opgeteld worden. Na dit kan het aantal bugs die opgelost zijn berekend worden door het verschil van deze twee waarden.

Relevantie: Door alle testfasen te vergelijken met elkaar wordt een beeld van de effectiviteit gecreëerd van deze fasen. De vergelijking gebeurt door alle bugs op te tellen en te sorteren per testfase die in de *tag* aanwezig is.

3.5.9 *Bugs gevonden tijdens Smoke* testen *[13][14]*

Definitie: Voor *smoke* testen is het nodig om alle primaire functies te testen en te controleren of deze werken. Net als bij 4.4.8 bepaalt deze waarde de kwaliteit van de testfase, in dit geval *smoke* testen

Berekening: De berekening gebeurt op dezelfde wijze als beschreven bij 4.4.8. Alleen de *tag* verandert naar *smoke* testen.

Relevantie: De relevantie is ook gelijk aan die van 4.4.8.

3.5.10 *Bugs gevonden tijdens Regression* testen *[7][13][14]*

Definitie: Net als bij 3.4.8 en 3.4.9 geldt dat deze waarde helpt bij het bepalen van de testfase kwaliteit en de effectiviteit ervan om bugs op te sporen.

Berekening: De berekening gebeurt op dezelfde wijze en de tag verandert naar *regression testen.*

Relevantie: Net als 3.4.8 en 3.4.9 geldt de relevantie om de verschillende testfasen met elkaar de vergelijken om de effectiviteit tussen de fasen te vergelijken.

3.5.11 *Bugs gevonden door de eindgebruikers*

Definitie: Bugs die gevonden worden door klanten worden als een *pink ticket* bijgehouden in de *backlog*. Indien klanten nieuwe bugs vinden kunnen deze in een lijst bewaard worden. De kwaliteit van het gehele productieproces wordt in kaart gebracht door een zicht te krijgen op het aantal bugs die gevonden worden door klanten.

Berekening: Door een evolutie van deze waarde te maken doorheen de tijd, wordt nagekeken welke vooruit- of achteruitgang er gemaakt is bij het werken aan de applicatie. Het aantal *pink tickets* wordt opgeteld per sprint of release. Daarnaast moet er rekening gehouden worden of er reeds een bug gelinkt is aan het *pink ticket*.

Relevantie: Een vergelijking tussen de bugs die gevonden zijn per testfase en de *pink tickets* biedt een extra inzicht in de kwaliteit van de testfasen.

3.6 Features

Features zijn de grote onderdelen van applicaties waaraan gewerkt wordt. Een *feature* wordt in kleinere onderdelen verdeeld om een beter overzicht te krijgen van de opdrachten die er aan gelinkt zijn. Daarnaast help de fragmentatie ook om te streven naar c*ontinuous delivery* doordat het eenvoudiger is om kleinere delen op korte tijd af te werken. Als streefdoel volgens het boek Accelerate [7], mag het werken aan een onderdeel van een *feature* niet langer duren dan één werkweek.

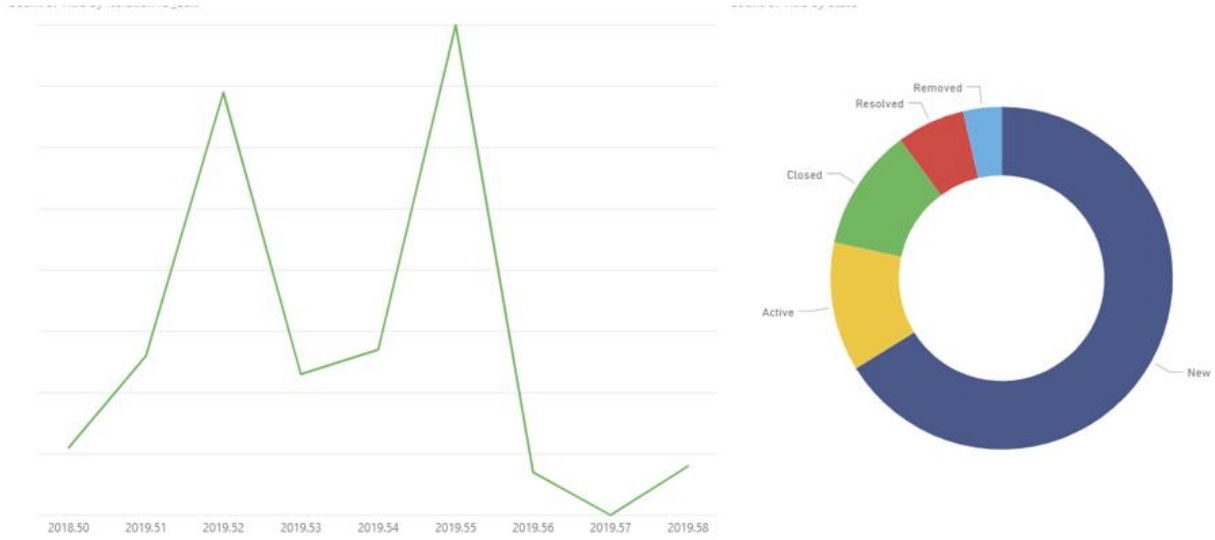
3.6.1 *Staat van Features*

Definitie: De staat van een feature is net als bij bugs het gegeven veld *State in* Azure DevOps. Die waarde helpt bij het inplannen van werk gelinkt aan features en biedt een overzicht van het werk dat al reeds gedaan is.

Berekening: De visualisatie gebeurt aan de hand van het optellen van alle *features* in de *backlog* en deze te sorteren per categorie die er aanwezig is in Azure DevOps (*Closed, Active, New, enz.*).

De visualisatie gebeurt door per sprint te kijken hoeveel features er in de *backlog* aanwezig zijn. Daarnaast kan met behulp van een cirkeldiagram gekeken worden naar de verhoudingen per status van features.

Voorbeeld:



Figuur 48: Features per sprint (links) en alle features aanwezig in de backlog (rechts).

3.7 Sprints

Om gestructureerd te werken binnen organisaties, wordt gebruik gemaakt van het sprintsysteem. Sprints zijn tijdsperioden van twee weken waarin een hoeveelheid werk ingepland wordt om af te werken. De hoeveelheid werk per *work item* wordt gebaseerd op het aantal gegeven *story points.*

3.7.1 *Status van Work Items*

Definitie: *Work items* bevatten een veld *State* waarin de status wordt bijgehouden van alle *items* die tijdens de sprint zijn ingepland. Door deze waarde te visualiseren, wordt een sprint overzicht bijgehouden tijdens de sprints. Daarnaast wordt op het einde van de sprints altijd een review gehouden. Een rapport met deze waarde in helpt bij het in kaart brengen van hoe goed of slecht een sprint verlopen is om het ingeplande werk af te werken.

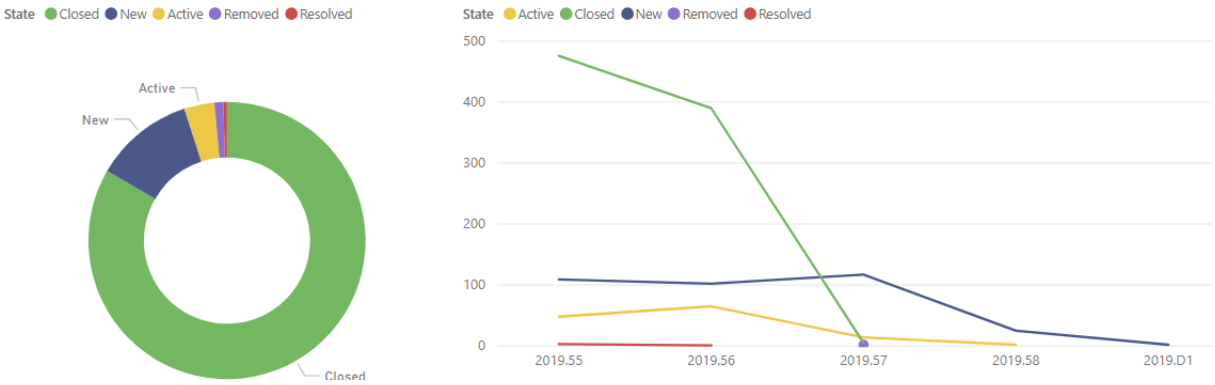
Berekening: Om een overzicht van de sprint te krijgen met deze waarde, worden alle *work items* opgeteld en gesorteerd op hun status.

De visualisatie wordt gedaan door een cirkeldiagram te gebruiken die de verschillende statussen weergeeft. Daarnaast kan er per sprint gekeken worden naar het aantal *work items*.

Een tweede mogelijkheid is de waarde per sprint te vergelijken in een lijngrafiek. Zo worden sprints met elkaar vergeleken en kan vooruit- of achteruitgang opgemerkt worden voor het afwerken van geplande *items.*

Relevantie: De werkballast en ernst van *work items* bieden een extra inzicht over de sprintplanning van vorige, huidige en eventueel toekomstige sprints.

Voorbeeld:



Figuur 49: Statussen van work items in de backlog (links) en vergeleken per sprint (rechts).

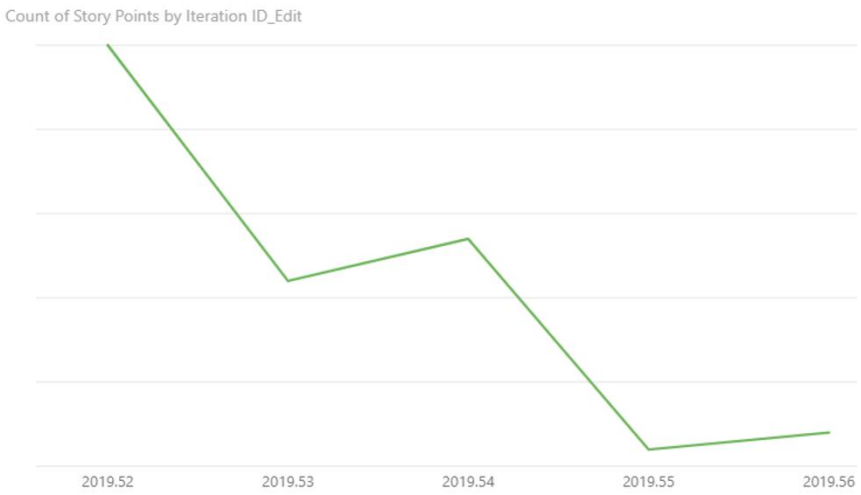
3.7.2 *Sprintgrootte*

Definitie: De grootte van sprints wordt bepaald door middel van het linken van *story points* aan *work items*. Elk team moet een hoeveelheid *story points* verdelen over de work items waaraan gewerkt wordt. Die helpen bij het geven van inzicht van de werklast per sprint.

Berekening: De items met *story points* in de *backlog* worden opgeteld. Daarna wordt gefilterd per team om een overzicht hiervan te krijgen.

Relevantie: De vergelijking van de sprints kan van belang zijn om de impact op het aantal bugs te vergelijken met het aantal story points die bepaald worden om af te werken tijdens een sprint. Zo kan er beslist worden om bijvoorbeeld meer tijd vrij te houden om bugs op te lossen. Dat heeft als gevolg dat er minder aan nieuwe functionaliteiten gewerkt kan worden.

Voorbeeld:



Figuur 50: Afgewerkte story points per sprint.

3.7.3 *Vooruitgang in werk [11]*

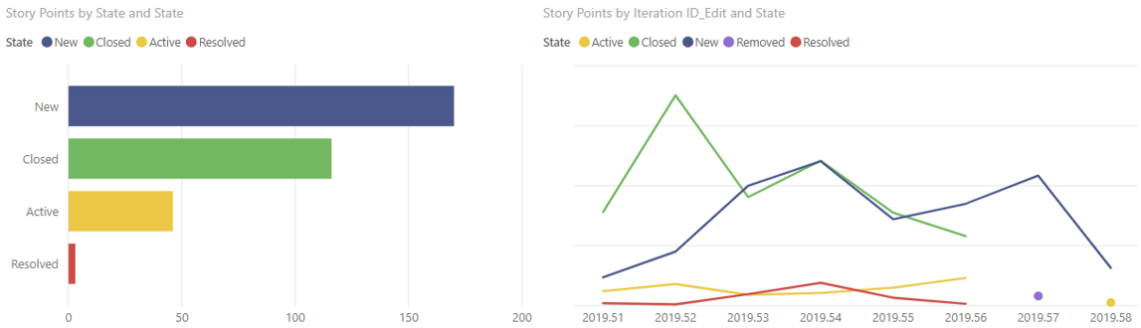
Definitie: De vooruitgang is een waarde die telkens gericht is op de actieve sprint. De waarde biedt een overzicht van hoeveel *story points* er reeds afgewerkt zijn en hoeveel er afgewerkt moeten zijn. Belangrijk is dat deze waarde enkel van belang is per team als een interne waarde.

Berekening: Voor elke sprint die er actief is, worden het aantal *story points* die gelinkt zijn aan een team opgeteld. Daarna wordt op vaste momenten *updates* gedaan van de *work items* die voor het veld *State* de waarde *Closed* bezitten. Als visualisatie kan er per sprint het aantal work items getoond worden per staat.

Daarnaast is het visualiseren per sprint ook mogelijk door de categorieën van staten waarin *work items* zich in bevinden de tonen per sprint. Dat biedt teams een inzicht in hoeveel items er afgewerkt worden per sprint.

Relevantie: Deze waarde is gefocust op de huidige situatie van het product. Die waarde kan relevant zijn met meerdere topics die van belang zijn om op het huidig moment te weten. Denk aan het aantal bugs die nog actief zijn en de ernst van deze bugs.

Voorbeeld:



Figuur 51: Work items gesorteerd per staat (links) en per staat weergegeven per sprint (rechts).

3.7.4 *Team Tevredenheid [7]*

Definitie: De tevredenheid van een team is gerelateerd aan hoe succesvol een nieuwe release is voor het team. Afhankelijk van welke problemen er nog steeds aanwezig zijn en welke opgelost zijn, verschilt de tevredenheid. Het is nuttig om bij te houden wat het effect is van de kwaliteitswaarden en de beslissingen die er mee gemaakt worden, op de tevredenheid van het bedrijf.

Berekening: De score kan gebaseerd zijn op een percentage dat de gemiddelde tevredenheid moet voorstellen van het bedrijf. De score wordt opgevraagd aan de hand van een enquête. Om dit zo efficiënt mogelijk te doen, wordt er gewerkt met een online poll die eenvoudig alle data verzamelt en kan doorsturen.

Relevantie: De tevredenheid van de teams wordt vergeleken met hoe tevreden de klanten zijn met de releases. Dat biedt een inzicht op het effect van wijzigingen aan de applicatie bij beide partijen.

3.8 Testfasen

Testfasen zijn gefocust op het opsporen van zoveel mogelijk bugs en andere fouten die in de applicatie aanwezig zijn. De kwaliteitswaarden onder dit thema zijn gelinkt aan de testfasen die aanwezig zijn in het productieproces. Er wordt een onderscheid gemaakt tussen *unit, smoke* en *regression* testen. Door deze waarden te gebruiken, wordt de kwaliteit bepaald van het productieproces en het effect van de verschillende testfasen..

3.8.1 *Aantal test cases: Unit testen [13]*

Definitie: Tijdens *unit testen* worden er *test cases* aangemaakt die fouten moeten opsporen in de geschreven onderdelen van de applicatie. Door een inzicht te creëren van deze waarde wordt bepaald wat de impact is van *unit testen* binnen de organisatie.

Berekening: Bij elke *build* zijn er een reeks van NUnit en Junit testen gelinkt die bewaard worden in een log op Azure DevOps. Omdat het niet mogelijk is deze om deze rechtstreeks op te vragen, wordt aanbevolen om de REST API te gebruiken die specifiek de *test cases* opvraagt per *build*. Daarna kunnen deze opgeteld worden per *build*.

Als visualisatie kan er een lijst opgebouwd worden die de *builds* toont die minder dan het gemiddelde aan *test cases* bevatten.

Relevantie: Om deze waarde beter in kaart te brengen is het van belang om 4.7.2 te betrekken bij de visualisatie. Niet elke *build* bevat hetzelfde aantal lijnen code. Er moet dus ook geëvalueerd worden op basis van de *code coverage.*

3.8.2 *Code Coverage [7][13]*

Definitie: Als evaluatie van de kwaliteit van *unit testen,* wordt met behulp van *code coverage* gewerkt. Dat is een waarde die aantoont hoeveel procent van de geschreven code nagekeken wordt met behulp van deze *test cases.* Door deze waarde te gebruiken, wordt een inzicht gegeven in de kwaliteit van de test fasen bij de applicatieontwikkeling.

Berekening: Deze waarde wordt enkel uitgelezen uit het platform waarin de code opgeslagen wordt.

Relevantie: De *code coverage* per *build* kan een verdiepend inzicht geven in de kwaliteit van de code.

3.8.3 *Resultaten: Smoke testen [13][14]*

Definitie: Bij elke *build* wordt tijdens *smoke testen* geëvalueerd of deze voldoet aan de eisen om door de gaan naar volgende ontwikkelingsfasen. Er worden rapporten bijgehouden van de geslaagde en gefaalde *test cases.* Door deze op te vragen, wordt geëvalueerd wat de impact is van *smoke testen* en de kwaliteit van de *builds.*

Berekening: Binnen de organisatie wordt gebruik gemaakt van het Robot Framework. Die houdt van elke *build* een rapport bij. Door met een API de rapporten uit te lezen en in een leesbaar formaat bij te houden, kan het aantal geslaagde en gefaalde cases bijgehouden worden. De visualisatie gebeurt door voor elke *build* te tonen wat het slaagpercentage is van elke *build*.

Relevantie: Per *build* worden er *build* aangemaakt in de *backlog* die ontdekt zijn tijdens de testfase. Als extra inzicht is het relevant om te kijken naar de bug waarden die specifiek gelinkt zijn aan deze *build*. Zo worden kritische problemen ontdekt met behulp van *smoke testen*.

3.8.4 *Resultaten: Regression Testen [13][14]*

Definitie: Nadat er wijzigingen aan de applicatie geslaagd zijn voor *smoke testen*, moet er nagekeken worden of deze een impact hebben op andere applicatieonderdelen. Indien er problemen gevonden worden tijdens deze fase, wordt bepaald of de *test case* slaagt of faalt. De resultaten van de *test cases* kunnen de kwaliteit is van de applicatie helpen bepalen.

Berekening: Bij manuele *regression testen* worden de resultaten bijgehouden in Azure DevOps. Door het aantal geslaagde en gefaalde *test cases* op te tellen, wordt een overzicht gegeven van het resultaat van *regression testen*. Dat resultaat kan weergegeven worden in een tekstvak voor een huidige *build*. Een andere mogelijkheid is om per *build* de *regression testen* resultaten in kaart te brengen met een lijngrafiek.

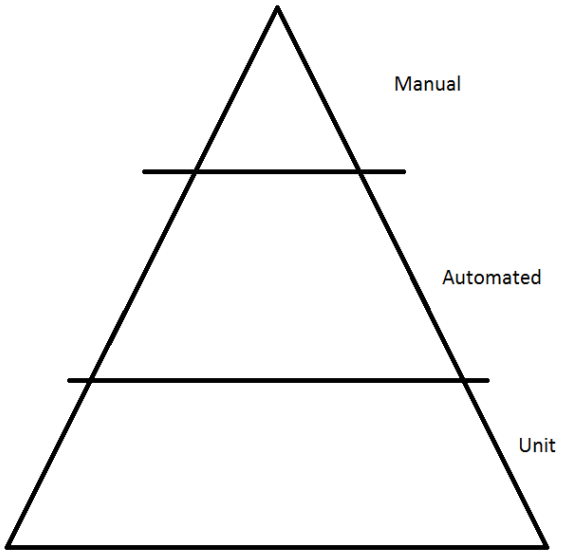
3.8.5 *Manuele en automatische testen [7]*

Definitie: Om manueel werk te minimaliseren en meer tijd vrij te maken voor ander werk, is het automatiseren van test cases belangrijk. Door bij te houden wat de verhouding is tussen manuele en automatische testen, kan er bepaald worden of er meer tijd vrij gemaakt moet worden om automatische testen te schrijven.

Berekening: In Azure DevOps zijn alle manuele test cases beschikbaar. Uit deze bron kan het aantal opgeteld worden. Voor de automatische test cases moet er een API gebruikt worden om het aantal cases uit het Robot Framework te halen en op te tellen.

De visualisatie om de verhouding tussen automatische en manuele testen te tonen, gebeurt aan de hand van een pyramide. De pyramide moet een idee geven van hoeveel testen er meer of minder per testfase moeten worden gemaakt.

Voorbeeld:



Figuur 52: Pyramidevorm per testfase.

3.9 Klantenfeedback

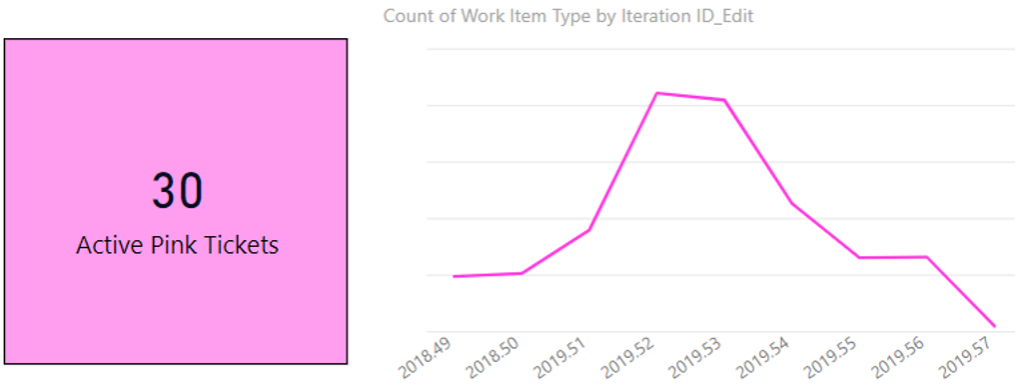
Klantenfeedback is het aantal *pink tickets* die aangemaakt worden in het *support*systeem van de organisatie. Daarbij zijn deze *pink tickets* gelinkt aan *work items* in de *backlog*. Klanten*tickets* (*pink tickets)* bestaan uit gemelde bugs, wensen naar nieuwe functionaliteiten, etc.

Definitie: Het aantal pink tickets is een opsomming van alle *bugs* die klanten gevonden hebben in de applicaties.

Berekening: Omdat *pink tickets* een samenvatting is van alle feedback die klanten geven en enkel de bugs voor dit onderzoek van belang zijn, moet er filter komen op de *pink tickets*. Dat wordt gedaan door een lijst op te stellen van *bugs* die een waarde hebben in het veld *Reference ID*. Indien dit niet ingevuld is, is het geen bug gerelateerd aan een *pink ticket*.

Relevantie: De ernst en statussen van *pink tickets* helpen bij een beter georganiseerde sprint indien er meer focus nodig is op *pink tickets.*

Voorbeeld:



Figuur 53: Aantal pink tickets voor actieve sprint (links) en per sprint (rechts).

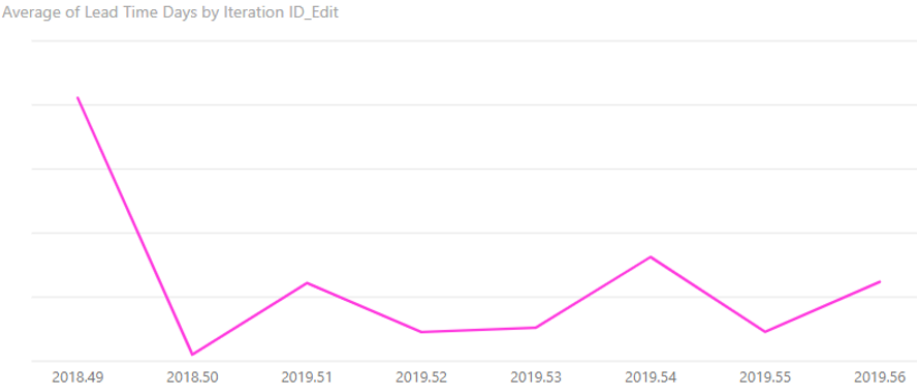
3.9.1 *Lead time van pink tickets [7][11]*

Definitie: Net als beschreven bij bugs, dient deze waarde om een zicht te krijgen op de tijd die nodig is om een *work item* af te handelen. Teams kunnen op basis van deze waarde ingrijpen indien de *lead time* te groot wordt.

Berekening: De berekening en visualisatie gebeuren op dezelfde wijze als bij 4.4.2. De enige verandering die er gebeurt is dat het *work item* type naar *pink tickets* geplaatst wordt.

Relevantie: Ook bij de *lead time* van *pink tickets* is het interessant om de vergelijking te maken met *cycle time*. Dat biedt een extra inzicht in de tijd die gespendeerd wordt om feedback van klanten te verwerken. Ook *reaction time* is relevant om te berekenen met behulp van *lead* en *cycle time.*

Voorbeeld:



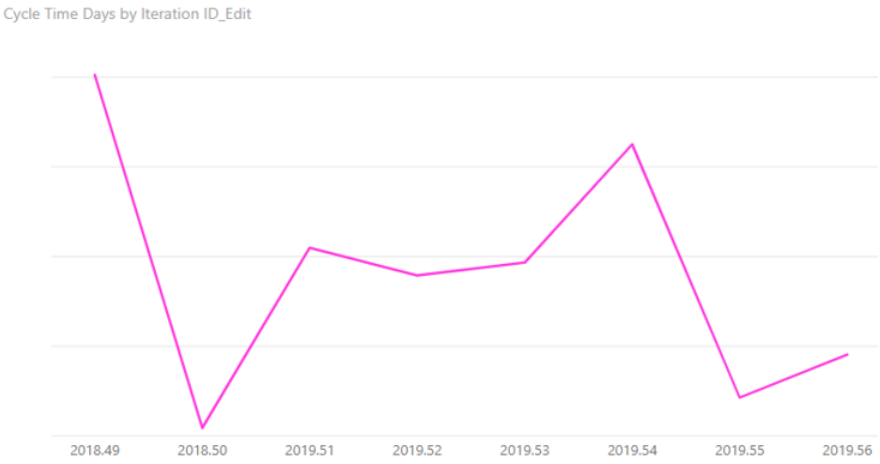
Figuur 54: Lead time van pink tickets.

3.9.2 *Cycle Time van Customer Feedback [7][11]*

Definitie: Net als bij *lead time*, geldt hier dezelfde uitleg waarom de waarde belangrijk is, maar dan enkel op de tijd die effectief gebruikt wordt om aan de *pink tickets* te werken.

Berekening: De berekening en visualisatie van deze waarde is identiek aan die van 3.4.3 (*Cycle time* van bugs). Enkel het *work item* type verandert naar *pink tickets.*

Voorbeeld:



Figuur 55: Cycle time van pink tickets.

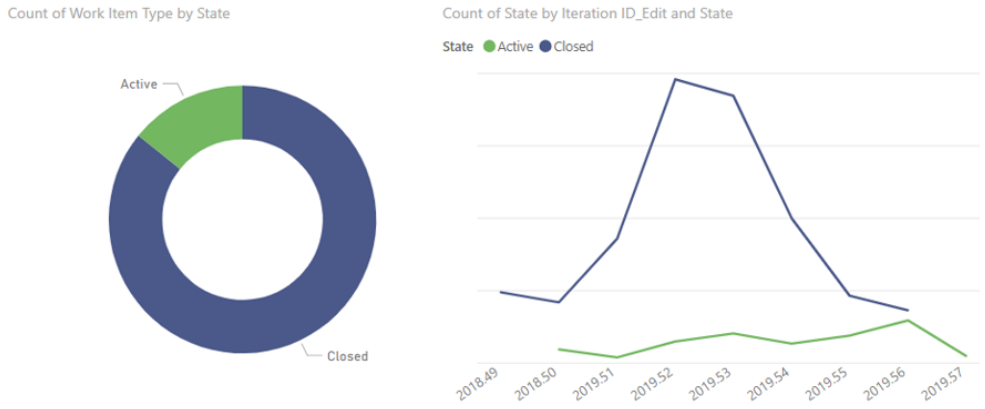
3.9.3 *Status van pink tickets*

Definitie: De staat waarin een *pink ticket* zich bevindt, wordt aangegeven door het veld *State*. Dat veld is een standaardwaarde die ingevuld wordt in Azure DevOps. De status wordt in de *backlog* bijgehouden als *New, Active, Closed* of *Resolved.*

Berekening: Om een inzicht te krijgen van het aantal bugs die gelinkt zijn aan *pink tickets*, moet er rekening gehouden met de aanwezigheid van een waarde in het veld *Reference ID* binnen Azure DevOps. Daarna kan het aantal bugs gesorteerd worden per status.

De visualisatie wordt gedaan met behulp van een cirkeldiagram om een algemeen overzicht te creëren. Indien er doorheen de tjid een overzicht moet zijn, is een lijngrafiek meer geschikt.

Voorbeeld:



Figuur 56: Status van pink tickets in actieve sprint (links) en als overzicht per sprint (rechts).

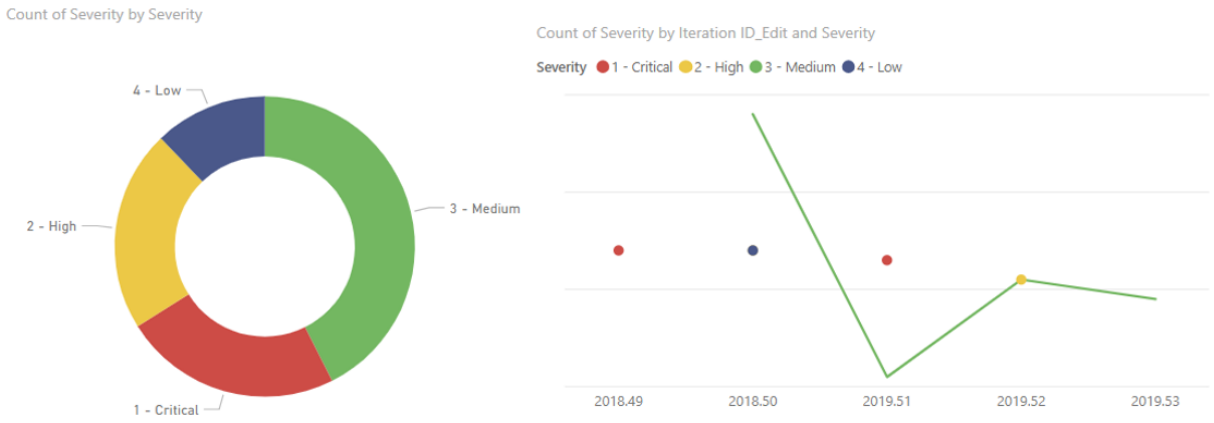
3.9.4 *Ernst van pink tickets*

Definitie: De ernst van een *pink ticket* wordt op dezelfde manier bepaald als bij bugs en andere *work item* types. Door deze waarde te visualiseren, wordt ingeschat hoeveel *pink tickets* een ernstige impact hebben op de applicatie.

Berekening: Om de visualisatie mogelijk te maken wordt een cirkeldiagram gebruikt om de huidige toestand van de applicatie weer te geven. Het aantal *pink tickets* wordt opgeteld en moet dan per categorie gesorteerd worden. Daarnaast kan een lijngrafiek per sprint aantonen hoe de applicatie er aan toe is in vergelijking met vorige sprints op vlak van klantenfeedback.

Relevantie: De ernst van *pink tickets* kan ook relevant zijn om een inzicht te krijgen per product onderdeel. Daarnaast kan een overzicht doorheen de tijd ook interessant zijn om een zicht te krijgen op de vooruitgang op het verwerken van klantenfeedback op basis van de ernst.

Voorbeeld:



Figuur 57: Ernst van pink tickets in actieve sprint (links) en per sprint (rechts).

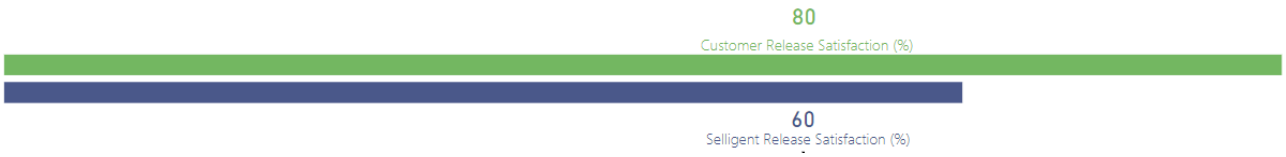
3.9.5 *Klanttevredenheid [7]*

Definitie: Om te begrijpen hoe klanten tegenover nieuwe *releases* staan, is het van belang om de feedback via hun kant te verwerken. Op basis van de feedback wordt afgeleid of er veranderingen teruggedraaid moeten worden om de tevredenheid te verbeteren.

Berekening: als berekening wordt een vragenformulier verzonden naar klanten waarbij er feedback gevraagd wordt over de tevredenheid. Op basis van de resultaten wordt een gemiddelde opgesteld van de tevredenheid van alle klanten. De visualisatie gebeurt door het percentage van de tevredenheid in een staafdiagram te verwerken en te vergelijken met de organisatietevredenheid van een release.

Relevantie: Een extra inzicht wordt gecreëerd door de teamtevredenheid bij de visualisatie te betrekken. Volgend voorbeeld is een fictieve visualisatie van hoe het resultaat er uit kan zien.

Voorbeeld:



Figuur 58: Fictieve visualisatie klant- en teamtevredenheid. (Dummy Data)

3.10 Technical *Debt*

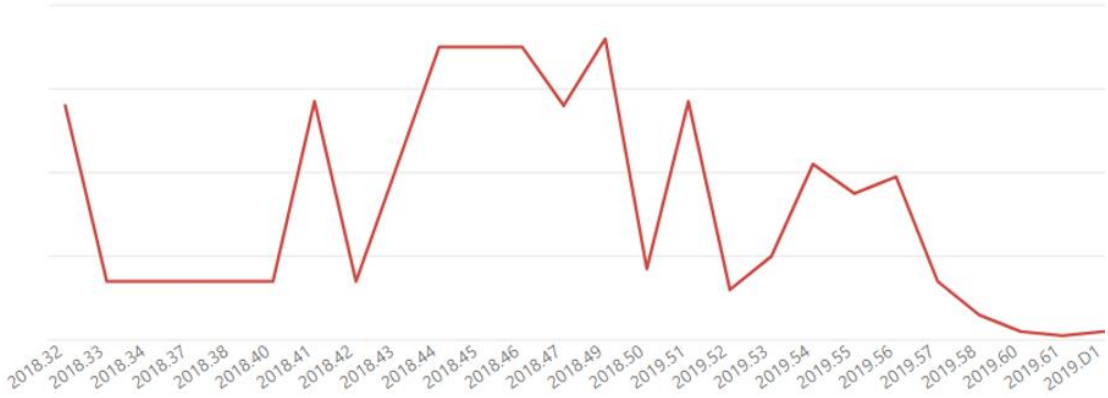
Technical Debt is de hoeveelheid werk die reeds met een korte-termijn oplossing weggewerkt is maar op langetermijn een probleem kan vormen. *Technical Debt* kan heel snel oplopen als er geen structurele manier van werken is om die niet uit de hand te doen laten lopen.

3.10.1 *Hoeveelheid Technical Debt*

Definitie: Het aantal *work items* van het type *technical debt.* Door deze waarde bij te houden wordt een overzicht gecreëerd van de hoeveelheid *technical debt.*

Berekening: Door het aantal *work items* op te tellen van het type *technical debt* in de *backlog* van Azure DevOps, kan de hoeveelheid berekend worden. De visualisatie gebeurt door per sprint te vergelijken hoeveel *technical debt* er nog bestaat in de *backlog.*

Voorbeeld:



Figuur 59: Hoeveelheid actieve technical debt opgenomen per sprint.

3.10.2 *Status van Technical Debt [3]*

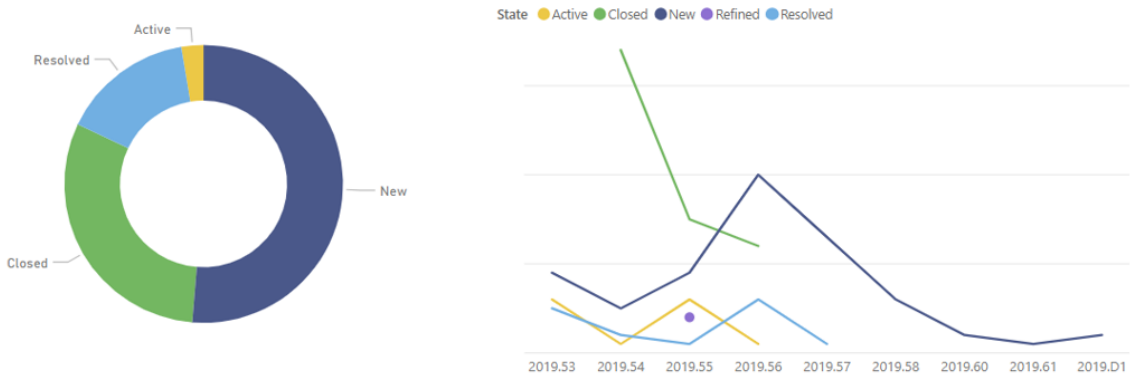
Definitie: De staat waarin een *technical debt* zich bevindt, wordt aangegeven door het veld *State*. Dat veld is een standaardwaarde die ingevuld wordt in Azure DevOps. Op basis van de statuscategorieën wordt bepaald hoeveel *technical debt* er reeds nog niet afgerond zijn. Door deze waarde de visualiseren, wordt de werkkwaliteit van actieve of vorige sprints bepaald. De staat wordt in de *backlog* bijgehouden als *New, Active, Closed* of *Resolved.*

Berekening: Om deze waarde te visualiseren moet alle *technical debt* per status gecategoriseerd worden. Afhankelijk van wat de gebruiker wil zien, kan er per sprint, release, maand, enz. doorheen de tijd gekeken worden naar de vooruitgang.

De visualisatie gebeurt met behulp van een cirkeldiagram om de waarde voor de huidige sprint te tonen. Voor een vergelijking tussen meerdere sprints is een lijngrafiek geschikt.

Relevantie: De ernst en werklast van bugs helpen bij het in kaart brengen van de werkkwaliteit om bugs op te lossen. Daarnaast is er een mogelijkheid om per productonderdeel te kijken naar de statussen zijn van de bugs.

Voorbeeld:



Figuur 60: Staat van technical debt voor huidige sprint (links) en per sprint (rechts).

3.10.3 *Ernst van Technical Debt*

Definitie: De ernst van een *technical debt* wordt op dezelfde manier bepaald als bij pink tickets en andere *work item* types. Door deze waarde te visualiseren, wordt ingeschat hoeveel *technical debt* een ernstige impact heeft op de applicatie.

Berekening: Om de visualisatie mogelijk te maken wordt een cirkeldiagram gebruikt om de huidige toestand van de applicatie weer te geven. De hoeveelheid *technical debt* wordt opgeteld en moet dan per categorie gesorteerd worden. Daarnaast kan een lijngrafiek per sprint aantonen hoe de applicatie er aan toe is in vergelijking met vorige sprints op vlak van klantenfeedback.

Relevantie: De ernst van *technical debt* kan ook relevant zijn om een inzicht te krijgen per product onderdeel. Daarnaast kan een overzicht doorheen de tijd ook interessant zijn om een zicht te krijgen op de vooruitgang van het verwerken van dit type *work item*.

Voorbeeld:

Figuur 61: Ernst van technical debt per sprint.

3.11 Werkkwaliteit

Naast de kwaliteit van het eindproduct is het ook van belang om de werkkwaliteit te optimaliseren. Door dit thema te verwerken in een *dashboard,* is er een onrechtstreekse impact op het eindproduct doordat er meer tijd gemaakt wordt voor essentiële opdrachten. Voor elk team is het belangrijk om een balans te vinden tussen proactief en reactief werk.

3.11.1 *Tijd gespendeerd aan proactief werk [17]*

Definitie: Proactief werk bestaat uit werk dat ingepland is tijdens sprints. Werk als ontwerpen en bouwen van *features, test cases* en andere applicatiestructuren worden geteld als proactief.

Berekening: De berekening gebeurt op basis van een ondervraging per team waarbij de leden van team het aantal uren moeten inschatten dat ze spenderen aan proactief werk. Als visualisatie wordt een verhouding gemaakt tussen proactief en reactief werk.

Relevantie: Het is van belang om de waarde 4.10.2 te betrekken bij de visualisatie om een idee te krijgen van de werkkwaliteit. Op basis van deze combinatie worden er veranderingen in werk afgesproken om de balans te krijgen zodat er meer proactief gewerkt wordt.

3.11.2 *Tijd gespendeerd aan reactief werk [17]*

Definitie: Reactief werk staat voor het werk dat onverwacht is en spontaan naar boven komt bij een probleem. Hieronder wordt verstaan: Problemen afhandelen, werk aan bugs en reageren op klantenfeedback.

Berekening: De berekening gebeurt op dezelfde wijze als bij proactief werk. De visualisatie gebeurt ook in combinatie met proactief werk.

3.12 Wat is Softwarekwaliteit?

Er kan gesteld worden dat volgens State of DevOps de grote spelers in de softwarewereld erg gefocust zijn op het minimaliseren van *lead*- en *cycle* time om problemen van klanten op te lossen. [4] Deze problemen kunnen binnen Selligent vertaald worden als bugs die gevonden worden door gebruikers (pink tickets). Om daarnaast te minimaliseren dat gebruikers zo min mogelijk bugs ondervinden, is het noodzakelijk om dan de kwaliteitswaarden gerelateerd aan bugs te visualiseren. Om de kwaliteit van de software te optimaliseren, is het belangrijk om de verschillende testfasen te optimaliseren. Daarom zijn de kwaliteitswaarden die gerelateerd zijn aan de testfasen ook belangrijk om in kaart te brengen tijdens het opbouwen van een kwaliteits*dashboard*.

3.13 State of DevOps

Om te begrijpen hoe de softwareontwikkeling kan verbeteren, wordt gekeken hoe andere grote spelers op de IT-markt het aanpakken. Dat wordt gedaan door de bron *Accelerate: State of DevOps* [4] te analyseren. In dit onderzoek wordt in kaart gebracht hoe grote, middelgrote en kleine organisaties hun productiviteit is op hunzelf en hoe ze verschillen met elkaar.

Grote spelers verdelen het werk dat verzet wordt tijdens de werkuren in vijf categorieën: Nieuw werk, ongepland werk, oplossen of controleren van beveiligingsbugs, werken aan defecten gevonden door gebruikers en klantenondersteuning. Voornamelijk de eerste vier categorieën zijn relevant om te linken aan softwarekwaliteit. Daarbij zou het nieuwe werk bij de grote organisaties 50% van de werktijd innemen. Hoe kleiner de organisaties, hoe minder tijd er aan nieuw werk gespendeerd wordt. De andere helft van de tijd wordt onder de andere vier categorieën verdeeld.

In het onderzoek wordt ook gekeken naar het manuele werk dat gedaan wordt en hoe de tijdsverdeling in dit onderwerp is. Manueel werk wordt verdeeld in vier categorieën: *Configuration Management*, Testen, *Deployments* en *Change Approvals*. De grootste spelers in het onderzoek spenderen het meeste manuele werk aan testen (10%) en *change approvals* (10%). Opvallend is dat middelgrote spelers tot de helft van het manuele werk spenderen aan manueel testen. Voor het eindwerk is Testen zeer relevant doordat er binnen Selligent net gewerkt wordt aan het automatiseren hiervan. De *Deployments* zijn ook relevant om de codeanalyses uit te voeren.

Er kan gesteld wordt dat ongepland werk gelijk staat aan 3.10.2 Reactief werk. Meer specifiek is de focus voor het oplossen van bugs erg belangrijk met beveiliging als hoogste prioriteit. Het werken aan defecten die klanten ondervinden kan binnen Selligent geplaatst worden onder *pink tickets*. Klantenondersteuning wordt minder onder softwarekwaliteit bepaald maar is zeker relevant om het proces achter pink tickets te optimaliseren.

Daarnaast is het van belang om een overzicht te krijgen van de verschillende testfasen om het manueel werk te verminderen. Ook is de codekwaliteit een belangrijke factor om de softwarekwaliteit en het proces er achter te bepalen.

3.14 **Conclusie**

Binnen softwareontwikkeling kan de kwaliteit van het productieproces in kaart gebracht worden door waarden op te meten die gerelateerd zijn aan de werkkwaliteit, de voorbereidingen en opvolgingen van sprints en het opvolgen en opsporen van bugs tijdens de verschillende stappen van de productie. Om de kwaliteit van het product op te volgen moet er in kaart gebracht worden hoe klanten tegenover het product staan, het optimaliseren en opvolgen van de verschillende testfasen, Het aantal gevonden bugs en de eigenschappen die gelinkt zijn aan bugs (Ernst, hoeveelheid werk om op te lossen, etc.), de kwaliteit van de *development* van het product en rekening houden met de hoeveelheid *technical debt*.

Van alle onderzochte meetbare waarden die helpen bij het bepalen van softwarekwaliteit, is er een selectie die specifiek helpt bij het opvolgen van zowel product- als proceskwaliteit binnen softwareontwikkeling. Allereerst moet de feedback die verkregen wordt door klanten in rekening worden gebracht. Daarnaast is het opvolgen en oplossen van de bugs die aanwezig zijn in de software en gemeld staan in de *backlog* van belang. Als laatste is het visualiseren van de resultaten van de testresultaten per testfase ook belangrijk om de softwarekwaliteit te bepalen. Dat is duidelijker te concluderen wanneer het kwaliteitsdashboard opgebouwd is in combinatie met de analyse van de verwerkte data.

Voor proceskwaliteit is zijn er meerdere domeinen van belang om een zicht op te krijgen. Wel is het voor product als proceskwaliteit van belang om een zicht te hebben om kwaliteitswaarden die aan bugs gerelateerd zijn. Daarnaast is het van belang om een goede opvolging van de sprintplanning te hebben waarbij de *product development* bijgehouden wordt. Dit wordt gedaan door de *work items* te visualiseren die gelinkt zijn aan een actieve sprint.

Daarnaast wordt het opvolgen van de werkkwaliteit gedaan door in kaart te brengen hoeveel reactief werk het team mee geconfronteerd wordt tijdens actieve sprints. Denk hierbij aan *technical debt* en problemen die sporadisch naar boven komen.

4 Visualisatiemogelijkheden

4.1 Inleiding

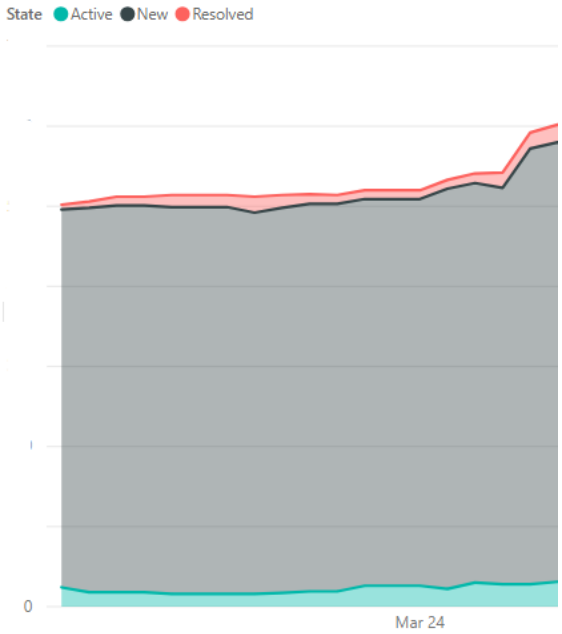
Om een beeld te krijgen van de gebruikte data, moet er nagedacht worden over de manier waarop dit gebeurt. Er zijn verschillende visualisatiemogelijkheden die elk op een eigen manier data kunnen visualiseren. Om softwarekwaliteit te visualiseren, wordt op verschillende manieren gewerkt om de kwaliteitswaarden te tonen. Dat hoofdstuk legt uit op welke manieren dit gebeurt en welke interpretaties mogelijk zijn bij de visualisaties. [5]

4.2 Historische Data

Voor het opvolgen van de productkwaliteit is het belangrijk om een inzicht te krijgen in hoe de progressie is doorheen de tijd. Om dit te doen wordt met historische data gewerkt die bepaalde patronen in de tijd kunnen aantonen. Op basis van deze patronen kan de organisatie proactief handelen om de trends die herhaaldelijk terugkomen.

Om *trends* in kaart te brengen, wordt er gewerkt met staafdiagrammen en lijngrafieken. Die helpen bij het visualiseren van uitzonderlijke momenten in de tijd. Daarnaast kan er met meerdere lijnen een vergelijking gemaakt worden tussen categorieën. Het voordeel van een lijngrafiek is dus het visualiseren van piekmomenten en om vergelijkingen te doen van categorieën over een tijdsperiode. Een nadeel van een lijngrafiek is dat het moeilijker is om exacte waarden af te lezen van de grafiek, voornamelijk wanneer er met meerdere categorieën gewerkt wordt. [8]

Een variant die het nadeel van de lijngrafiek oplost is een *area plot*. Die biedt een overzicht van een geheel door de verschillende categorieën bij elkaar te betrekken. Een nadeel van deze variant is dat het niet gebruikt kan worden om categorieën met elkaar te vergelijken.



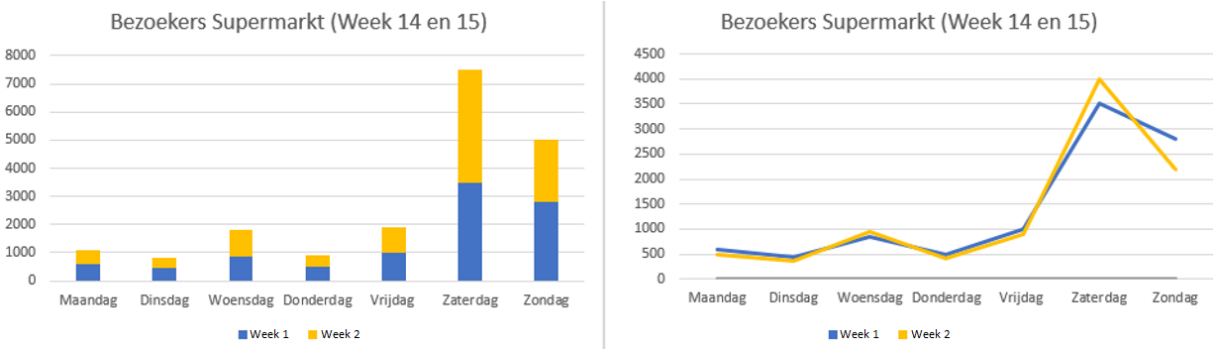
Figuur 62: Area plot voorbeeld

Een tweede manier om trends te visualiseren is met behulp van staafdiagrammen. Meer specifiek is het gebruik van gestapelde staafdiagrammen voor *datasets* een goede visualisatiemethode.

Door dit diagram te gebruiken wordt een totaalbeeld gecreëerd van de dataset en kan deze per categorie of over de tijd heen vergeleken worden. [8]

Als voorbeeld van de uitgelegde theorie:

Supermarkten willen weten hoeveel klanten er per weekdag binnenkomen. Dit moet hen helpen om proactief te kunnen handelen op de drukte. Wanneer er met een gestapelde staafdiagram het aantal klanten van twee weken gevisualiseerd wordt, kan aangetoond worden dat over het algemeen de weekenddagen de drukste zijn. Via een lijngrafiek en per week te kijken, kan er afgeleid worden dat in week 2 zaterdag drukker was dan week 1. Daarnaast blijkt ook dat zondag van week 1 drukker was dan die van week 2.

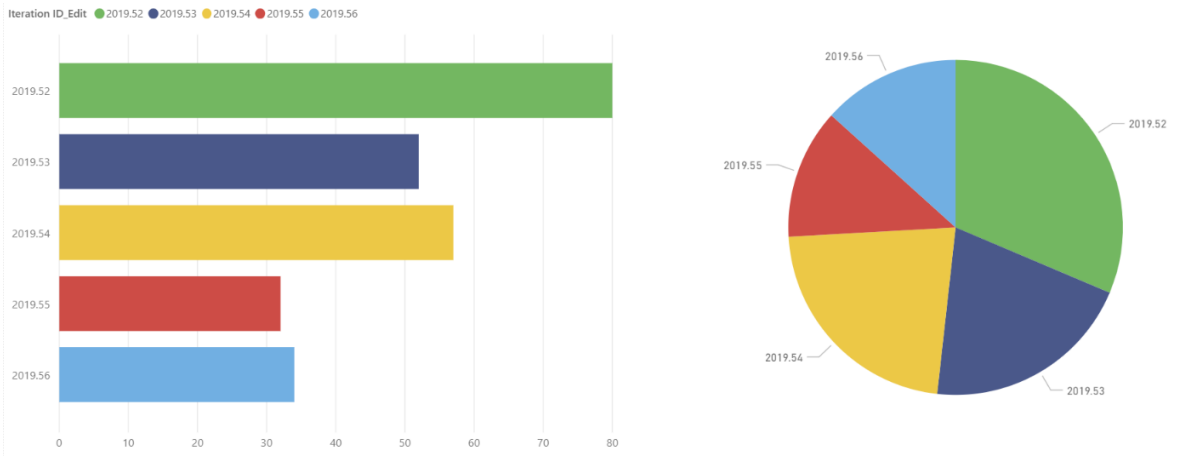


Figuur 23: Fictief voorbeeld Historische data.

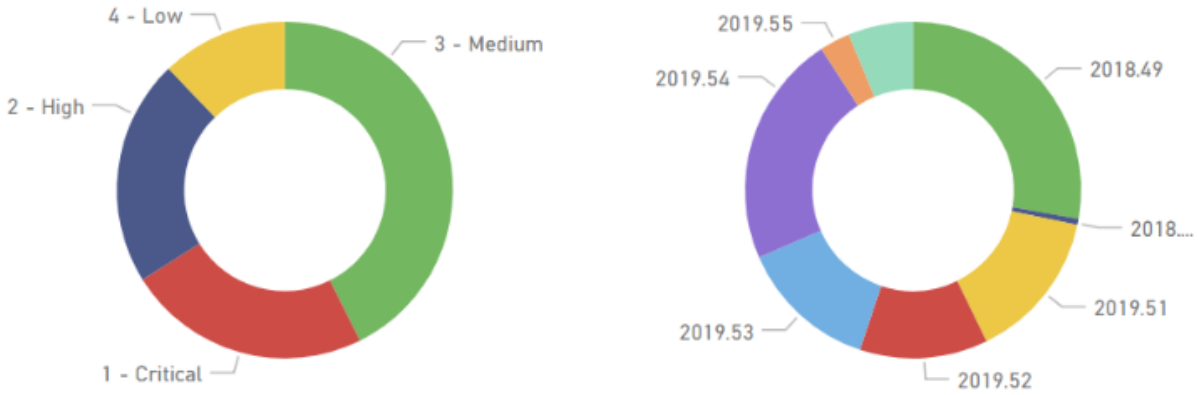
4.3 Vergelijken en Rangschikken

Als tijd geen relevante factor is om categorieën te vergelijken, dan kan er een rangschikking gebeuren per categorie om een overzicht te krijgen. Om een visualisatie van deze *dataset* te hebben, wordt er gewerkt met staafdiagrammen en cirkeldiagrammen.
Een staafdiagram biedt een visualisatie die eenvoudig een vergelijking per categorie kan tonen. Daarbij kan er ook een duidelijk onderscheid per categorie gemaakt worden op basis van hun totaalwaarden. [8][18]

Een cirkeldiagram heeft als nadeel dat de exacte waarden moeilijker af te lezen zijn. Wanneer bepaalde waarden in zeer kleine waarde verschillen, is het moeilijk om te onderscheiden welke een grotere waarde heeft dan de ander. Het voordeel van een cirkeldiagram is dat er met weinig informatie snel en duidelijk een beeld gecreëerd kan worden. Dat beeld kan enkel gemaakt worden wanneer de fragmentatie van de categorieën beperkt blijft. Indien er een te grote fragmentatie is van de categorieën, is het aangeraden om een selectie te maken van de belangrijkste categorieën om te visualiseren. [18]



Figuur 24: Vergelijking tussen aantal items per sprint.



Figuur 63: Voorbeeld van segmentatie.

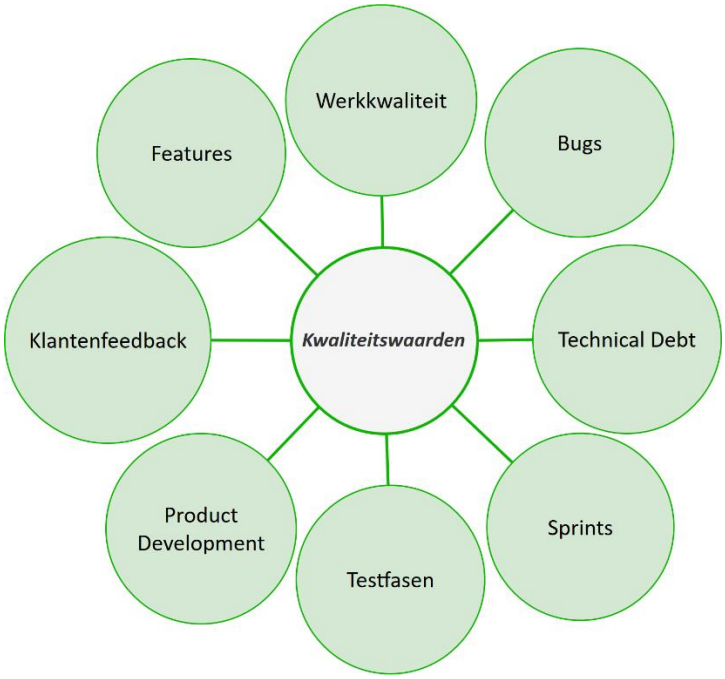
4.4 Conclusie

Lijngrafieken zijn het meest geschikt wanneer er gewerkt wordt met historische data om *trends* doorheen de tijd te kunnen visualiseren. Daarnaast kunnen staafdiagrammen ook een oplossing bieden om trends te visualiseren en daarbij exacte waarden zichtbaarder te maken in vergelijking met lijngrafieken.

Indien tijd geen factor is, kunnen er vergelijkingen gemaakt worden tussen verschillende categorieën. Dat gebeurt door ook staafdiagrammen te gebruiken om exacte vergelijkingen te doen. Indien het voldoende is om op een snelle manier categorieën te vergelijken en exacte waarden zijn niet van belang, dan is een cirkeldiagram voldoende om te gebruiken.

III. Conclusie

1 Meetbare waarden softwarekwaliteit

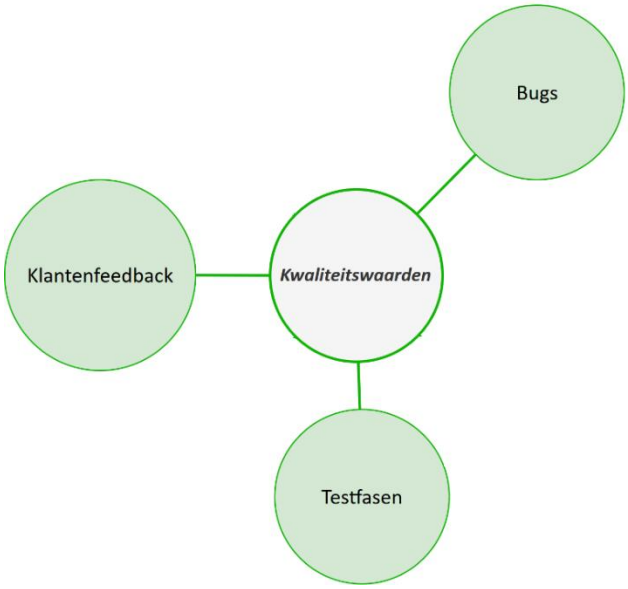


Figuur 64: Onderverdeling kwaliteitswaarden

De kwaliteit van het product en productieproces binnen softwareontwikkeling wordt in kaart gebracht door waarden te zoeken die geplaatst kunnen worden onder acht onderdelen. De onderdelen bestaan uit: Het bijhouden van en rekening houden met klantenfeedback, een lijst van de *features* en elementen die eraan gelinkt zijn, de werkkwaliteit van de *engineers* tijdens de werkuren, het bijhouden van de bugs en zijn elementen, het opvolgen van *technical debt*, de sprintplanning en alle voorbereidingen gerelateerd aan sprintplanning en de testresultaten en kwaliteit van elke testfase.

2 Selectie voor meten product- en proceskwaliteit

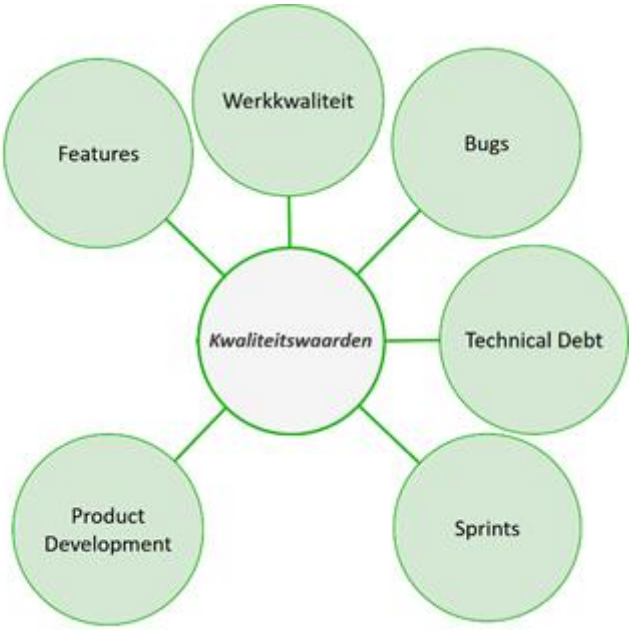
2.1 Productkwaliteit



Figuur 65: Selectie kwaliteitswaarden productkwaliteit

Na het opbouwen van een lijst met alle mogelijke meetbare kwaliteitswaarden is er een selectie gemaakt van waarden die zowel product- als proceskwaliteit kunnen helpen visualiseren. Van de acht onderdelen zijn er daar drie van geselecteerd. De primaire focus om de kwaliteit te visualiseren is de verzameling aan klantenfeedback en bugs die door klanten gemeld worden. Als tweede belangrijke onderdeel is het van belang om alle bugs en de waarden die gelinkt zijn aan bugs in kaart te brengen. Om te evalueren of het verwerken van de klantenfeedback en de bugs succesvol gelukt zijn, zijn testfasen en de resultaten er van het meest geschikt.

2.2 Proceskwaliteit



Figuur 66: Selectie kwaliteitswaarden proceskwaliteit

Voor proceskwaliteit is zijn er meerdere domeinen van belang om een zicht op te krijgen. Wel is het voor product als proceskwaliteit van belang om een zicht te hebben om kwaliteitswaarden die aan bugs gerelateerd zijn. Daarnaast is het van belang om een goede opvolging van de sprintplanning te hebben waarbij de *product development* bijgehouden wordt. Dit wordt gedaan door de *work items* te visualiseren die gelinkt zijn aan een actieve sprint.

Voor proceskwaliteit is zijn er meerdere domeinen van belang om een zicht op te krijgen. Wel is het voor product als proceskwaliteit van belang om een zicht te hebben om kwaliteitswaarden die aan bugs gerelateerd zijn. Daarnaast is het van belang om een goede opvolging van de sprintplanning te hebben waarbij de *product development* bijgehouden wordt. Dit wordt gedaan door de *work items* te visualiseren die gelinkt zijn aan een actieve sprint.

Daarnaast wordt het opvolgen van de werkkwaliteit gedaan door in kaart te brengen hoeveel reactief werk het team mee geconfronteerd wordt tijdens actieve sprints. Denk hierbij aan *technical debt* en problemen die sporadisch naar boven komen.

3 Blootleggen van anomalieën

Als conclusie op Hoofdstuk 4: Vusalisatiemogelijkheden, zijn er twee manieren die voornamelijk gebruikt worden om waarden zichtbaar te maken zijn: werken met trends en het rangschikken van categorieën. Indien het belangrijk is om trends doorheen de tijd bloot te leggen, dan zijn lijngrafieken en staafdiagrammen geschikt. Lijngrafieken kunnen trends van verschillende categorieën duidelijk visualiseren. Staafdiagrammen zijn bruikbaarder indien exacte waarden aflezen belangrijk is.

Voor het rangschikken van categorieën zijn staafdiagrammen geschikt om de exacte vergelijkingen te maken. Indien het belangrijk is om op een snelle en eenvoudige manier verschillen tussen categorieën te zien, dan is een cirkeldiagram aan te raden om te gebruiken.

4 Vergelijking visualisatietools

	Jupyter Notebook	Azure DevOps	Power BI
Instapniveau	✦	✦✦	✦✦✦
Voorbereiding	✦	✦✦	✦✦✦
Data *Cleaning*	✦✦✦	✦	✦✦
Updaten *Datasets*	✦✦	✦✦✦	✦
Live Data	✦	✦✦✦	✦✦
Visualisatiemogelijkheden	✦✦✦	✦	✦✦
Opbouw *Dashboards*	✦	✦✦	✦✦✦

Uit de toolvergelijking kan gesteld worden dat de drie onderzochte tools elk hun sterktes en zwaktes hebben. Afhankelijk van wat een persoon, team of de organisatie wilt hebben van resultaat, verschilt de selectie van de tool.

Indien het erg belangrijk is om uitgebreid en verdiepend data te visualiseren, is Jupyter Notebook met Pandas het meest geschikt. Via deze methode kan er een rapport opgebouwd worden die naderhand gebruikt kan worden als document in andere bestanden. Het instapniveau om op deze manier te werken ligt hoger dan de andere tools. Daarnaast is er geen mogelijkheid om met '*live*-data' te werken.

Azure DevOps is naast een databron ook een visualisatiemethode. Indien het belangrijk is om met *datasets* te werken die constant moeten geüpdatet worden, dan is de functie binnen Azure DevOps het meest geschikt. Het nadeel is een beperkte set aan visualisatiemogelijkheden en is data *cleaning* afhankelijk van hoe gestructureerd de *backlog* opgebouwd is.

Power BI biedt een combinatie aan van Azure DevOps (visualisatie) en Jupyter Notebook. Zo wordt op een eenvoudige manier met een uitgebreider aanbod data gevisualiseerd. Het instapniveau is zeer laag doordat het met een *interface* werkt die soortgelijk is aan Windows Office applicaties. Een nadeel is dat het updaten van *datasets* erg lang duurt afhankelijk van de hoeveelheid en grootte van de *datasets*.

5 Status van Selligent productkwaliteit

Besproken in interne versie eindwerk.

6 Status van Selligent proceskwaliteit

Besproken in interne versie eindwerk.

Bibliografie

[1] P. Caron, Creating and Using Quality Metrics in Software Delivery – Part 1, https://www.linkedin.com/pulse/creating-using-quality-metrics-software-delivery-part-peter-caron. [Geopend 27 februari 2019]

[2] P. Caron, Creating and Using Quality Metrics in Software Delivery – Part 2, https://www.linkedin.com/pulse/creating-using-quality-metrics-software-delivery-part-peter-caron-1f/. [Geopend 27 februari 2019]

[3] P. Caron, Creating and Using Quality Metrics in Software Delivery – Part 3, https://www.linkedin.com/pulse/creating-using-quality-metrics-software-delivery-part-peter-caron-2f/. [Geopend 27 februari 2019]

[4] G. Miranda, X. Velasquev, „Accelerate: State of Devops, Strategies for a New Economy´´,2018

[5] Data Visualization beginner's guide, https://www.tableau.com/learn/articles/data-visualization. [Geopend 1 maart 2019].

[6] W. Filho, What is a deployment pipeline and how does it help software development teams?, https://medium.com/the-making-of-appear-in/what-is-a-deployment-pipeline-and-how-it-helps-software-development-teams-6cb29917ceea. [Geopend 1 maart 2019].

[7] N. Forsgren, J. Humble, G. Kim, „The Science of Lean Software and DevOps: Accelerate", 2018

[8] Tableau, „Visual Analysis Best Practises", Available: https://www.tableau.com/sites/default/files/media/whitepaper_visual-analysis-guidebook_0.pdf [Geopend 2 april 2019]

[9] Smoke Testen, http://softwaretestenfundamentals.com/smoke-testen/. [Geopend 11 april 2019].

[10] API Testen Tutorial, https://www.guru99.com/api-testen.html. [Geopend 11 april 2019].

[11] S. A. Lowe, 9 Metrics That Can Make a Difference in Todays Software Development Teams, https://techbeacon.com/app-dev-testen/9-metrics-can-make-difference-todays-software-development-teams [Geopend 11 april 2019].

[12] Essential QA Metrics, https://rainforestqa.com/wp-content/uploads/2018/02/6-metrics-infographic-01-1.png [Geopend 11 april 2019].

[13] S. Seela, R. Yackel, 64 Essential Testen Metrics for Measuring Quality Assurance Success, https://www.qasymphony.com/blog/64-test-metrics/. [Geopend 11 april 2019]

[14] S. Raju, „Measurement and Analysis of Test Suite Volume Metrics for Regression Testion" Available: https://pdfs.semanticscholar.org/d193/48591637df97f00334f7e92299cfcb0275bc.pdf [Geopend 11 april 2019].

[15] Cycle, Lead and Reaction Time, https://airfreshener.club/quotes/are-cycle-what-times.html. [Geopend 10 april 2019]

[16] Azure DevOps WIQL Language, https://docs.microsoft.com/en-us/azure/devops/boards/queries/wiql-syntax?view=azure-devops [Geopend 10 maart 2019].

[17] A. McPeak, Making Your Site Pixel Perfect with Visual Regression Testen, 2018, https://crossbrowsertesten.com/blog/visual-testen/visual-regression-testen/. [Geopend 11 april 2019]

[18] https://www.tableau.com/sites/default/files/whitepapers/which_chart_or_graph_v2.pdf [Geopend 1 maart 2019].

[19] Plot.ly , https://plot.ly/. [Geopend 27 april 2019]

[20] Power BI, https://powerbi.microsoft.com/en-us/. [Geopend 29 april 2019]

[21] Azure DevOps, https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops. [Geopend 29 april 2019]

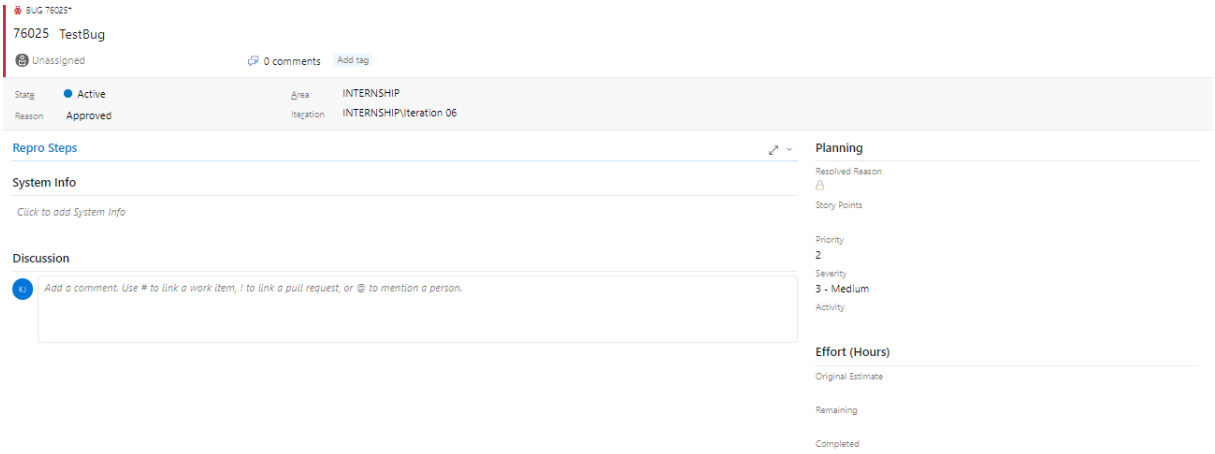
[22] Duplicate Code, https://refactoring.guru/smells/duplicate-code. [Geopend 20 mei 2019]

[23] Azure DevOps REST API, https://docs.microsoft.com/nl-nl/rest/api/azure/devops/?view=azure-devops-rest-5.1. [Geopend 28 mei 2019]

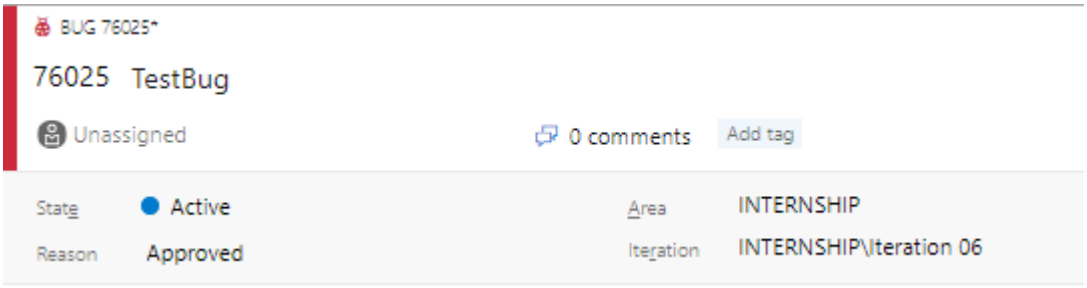
Bijlagen

A. Overzicht van een *Work Item* in Azure DevOps

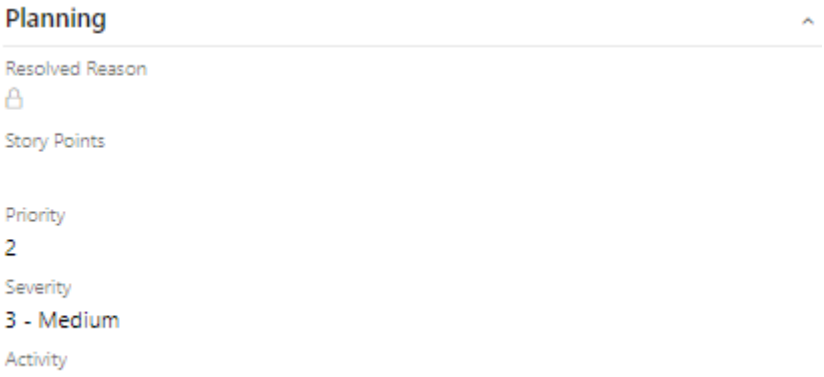
A. Overzicht van een *Work Item* in Azure DevOps



Bij het openen van een *work item* wordt bovenstaand scherm geopend.



Bovenaan het menu kunnen de *State*, Product *Area (Area)* en de gelinkt sprint *(Iteration)* teruggevonden worden.



Rechts kunnen de Prioriteit (*Priority*) en ernst (*Severity*) teruggevonden worden.