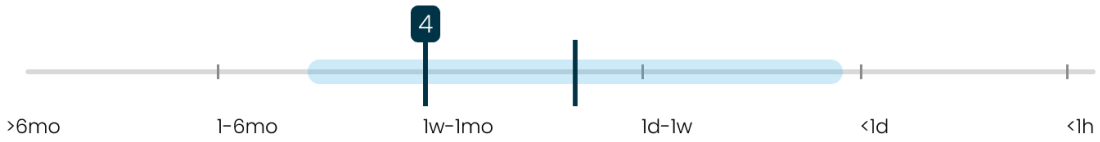
DORA-METRIEKEN.

Example Report DORA-Analyse.

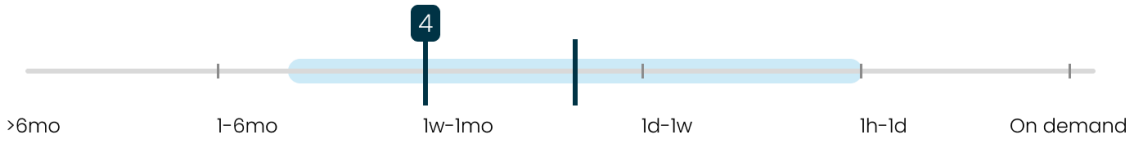
AVISI

Team Vecht DORA analysis.

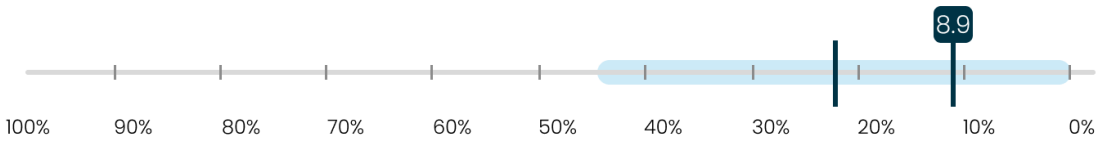
Lead time
One month to one week



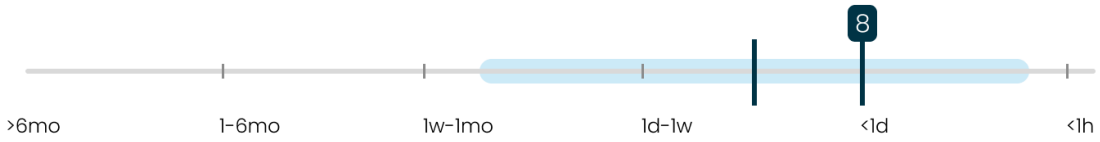
Deploy frequency
One month to one week



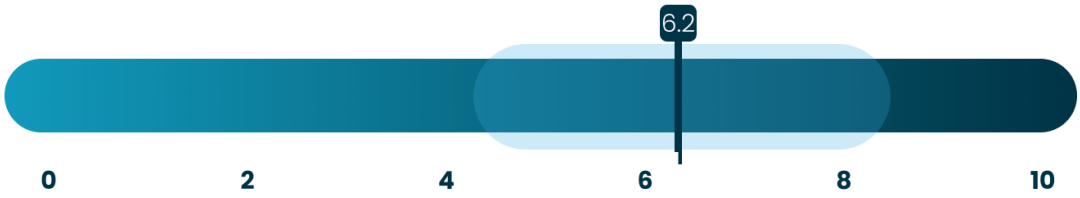
Change fail rate
11% of changes fail



Deployment recovery time
Less than a day



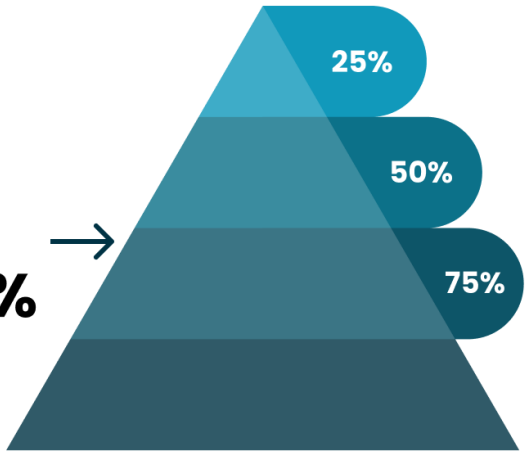
6.2



Quick wins:

- Apache DevLake in gebruik nemen
- Jira issues concreet labelen
- Parallele testingomgevingen opzetten
- Uitbreiden van automatische testsuite

Vecht top 55%



+ Complete teambrede analyse van moeilijke technische code gebieden doormiddel van reviewstrategie
+ Snelle ticketing en refinement methodiek die goed aansluit aan de benodigheden van het development team
– Veel verlies in het huidige acceptatieproces
– Weinig inzicht in de huidige performance van het development team
– Weinig controle over performance dashboards en het deployen van software

Table of Contents.

Management Samenvatting 4
Huidige Situatie 4
DORA Metrics 4
Aanbevelingen 5
Roadmap 5
Waardeberekening 5
Conclusie 5
Inleiding 6
DORA 6
Het doel 7
Huidige ontwikkelproces, wensen en uitdagingen 8
Resultaten gap analyse 8
Plan 8
Refinement 9
Development 9
Testing 10
Deployment 11
Monitoring 11
Huidige status DORA metrieken 12
Oplossingen 14
US-1 17
Meer inzicht krijgen in Jira-tickets 17
Dashboarding met Grafana 18
US-2 20
DORA-metrieken inzichtelijk maken met Apache Devlake 20
Knowledge-silo's identificeren met CodeScene 21
US-3 24
Pair programming 24
Mob programming 25

Mob testing 25
US-4 26
Automatische Security Testing met GitLab 26
GitLab dependency Scanning 28
GitLab license Scanning 29
GitLab secret Detection 29
US-5 31
(Scaled) Trunk Based Development 31
GitLab Deployment 33
Feature Flags 33
Canary Deployment 34
US-6 36
Monitoring m.b.v. Dynatrace 36
Dashboarding met Grafana 36
Roadmap 38
Kostenoverzicht 47
Licenties 47
Waardeberekening 49
Return on Investment & Payback Period 49
Cost of Unnecessary Rework Avoided Per Year 50
Potential Revenue from Reinvestment 51
Cost of Downtime 52
Potential Return 53
Verwachtingen 54

Afkorting	Betekenis
SDLC	Software Development Lifecycle
DORA	DevOps Research and Assessment
ROI	Return On Investment

Management Samenvatting

Dit rapport presenteert de resultaten van de DORA Gap Analyse sessies die zijn uitgevoerd door Ace en GitLab Expert Services. Het doel van deze sessies was om de huidige Software Development Lifecycle (SDLC) van Ace te evalueren en verbeterpunten te identificeren, met de focus op het verbeteren van processen, technieken en software.

Huidige Situatie

De huidige SDLC van Ace is onderverdeeld in zeven fases: Plan, Refinement, Development, Testing, Acceptatie, Deployment en Monitoring. Tijdens de analyse zijn verschillende uitdagingen en wensen van het team naar voren gekomen, waaronder:

- Onvoldoende inzicht in de status van tickets en de voortgang van het ontwikkelproces.
- Knowledge-silo's binnen het team, wat leidt tot afhankelijkheid van individuele teamleden.
- Inefficiënties in het test- en acceptatieproces, wat de doorlooptijd en kwaliteit van de software beïnvloedt.

DORA Metrics

De DORA-metrics, zijn metrieken die opgesteld zijn door de DORA research group, wat tegenwoordig onderdeel is van Google Cloud. Deze metrieken zijn er om de prestaties van DevOps-teams meten en worden gehanteerd als industriestandaard. Ace maakte nog geen gebruik van deze metrieken en is beoordeeld op basis van deze metrieken. De huidige scores van Ace zijn onder gemiddeld, wat aangeeft dat er ruimte voor verbetering is. De vier DORA-metrics zijn:

1. **Lead Time**

2. **Deploy Frequency**
3. **Change Fail Percentage**
4. **Failed Deployment Recovery Time**

Aanbevelingen

Op basis van de bevindingen zijn er zeven user stories geformuleerd, elk met bijbehorende oplossingen. Enkele belangrijke aanbevelingen zijn:

- **Inzicht in Jira-tickets**: Verbeteren van ticketbeheer door gebruik te maken van labels en dashboards.
- **Automatisering van tests**: Implementeren van automatische security tests en andere geautomatiseerde testmethoden binnen GitLab.
- **Pair en Mob Programming**: Stimuleren van kennisdeling en samenwerking binnen het team om de kwaliteit van de code te verbeteren en reviewtijden te verkorten.
- **Feature Flags en Canary Deployments**: Invoeren van feature flags en canary deployments om de releaseprocessen te stroomlijnen en risico's te minimaliseren.
- **Monitoring m.b.v. Dynatrace en Grafana:** Dynatrace gebruiken voor uitgebreide monitoring van de applicatie, terwijl Grafana wordt ingezet voor het bouwen van dashboards die inzicht geven in de productieomgeving en DORA-metrics.

Roadmap

De roadmap voor implementatie is verdeeld in drie fasen, met een totale tijdsbesteding van 352 uur. Elke fase richt zich op specifieke verbeterpunten en het behalen van meetbare resultaten in de DORA-metrics.

Waardeberekening

De verwachte ROI van de voorgestelde verbeteringen is significant, met een geschatte jaarlijkse besparing van €99.414 door het verminderen van onnodige herwerk, het verhogen van de efficiëntie en het verbeteren van de kwaliteit van de software.

Conclusie

De voorgestelde verbeteringen in de SDLC van Ace verhogen de efficiëntie en kwaliteit van de SDLC, wat zal leiden tot een verhoogde klanttevredenheid.

Inleiding

Dit document is een rapport van de DORA Gap Analyse sessies die Ace (klant) en GitLab Expert Services (consultant) samen hebben gehouden, om te kijken naar de huidige Software Development Lifecycle (SDLC) van Ace en hoe deze verbeterd kan worden met nieuwe technieken, processen en software. Aanwezig waren John Doe van Ace en Jane Doe van GitLab Expert Services.

DORA

DORA, wat staat voor DevOps Resource & Assessment, is een langdurig doorlopend onderzoek dat wordt uitgevoerd door een team binnen Google. Door het DORA-team wordt er ieder jaar een rapport uitgebracht, dat het "DORA Accelerate State of DevOps report" heet. Het onderzoek focust op hoe DevOps-teams binnen de gehele of een specifieke industrie presteren. Dit wordt onderzocht door middel van het uitvragen bij allerlei verschillende bedrijven en partijen. DORA focust over het algemeen op 4 metrieken. Wat elke metriek inhoudt en de meerwaarde voor een bedrijf die hieraan gerelateerd is, wordt hieronder genoemd:

- **Lead Time:** Voor de primaire applicatie of dienst waar je aan werkt: wat is de doorlooptijd voor wijzigingen (dat wil zeggen, hoe lang duurt het om van gecommitte code naar succesvol draaiende code in productie te gaan)?
    - Sneller features releasen, betekent dat tijd efficiënter gespendeerd wordt. Indien een team efficiënter werkt, is de verkregen waarde tegenover de geïnvesteerde tijd hoger en is de prestatie van het team als geheel beter.
- **Deploy Frequency:** Voor de primaire applicatie of dienst waar je aan werkt: hoe vaak voert jouw organisatie code uit naar productie of geeft deze vrij aan eindgebruikers?
    - Een hoge deploy frequency betekent dat het team nergens op hoeft te wachten om te deployen en dat het dus in feite op ieder moment gedaan kan worden. Dit betekent niet dat er iedere dag gedeployd moet worden, maar dat het dus wel altijd mogelijk is en developers dus hun tijd kunnen spenderen aan het ontwikkelen van nieuwe features.
- **Change Fail Percentage:** Voor de primaire applicatie of dienst waar je aan werkt: welk percentage van de wijzigingen in productie of vrijgegeven aan gebruikers leidt tot verminderde dienstverlening (bijvoorbeeld tot verstoring of uitval van de dienst) en vereist vervolgens herstel (bijvoorbeeld een hotfix, rollback, fix forward of patch)?
    - Een lager percentage brekende changes betekent dat het team minder tijd kwijt is met het verwerken van bugfixes. Deze tijd kan beter gespendeerd worden aan het ontwikkelen van nieuwe features.

- **Failed Deployment Recovery Time:** Voor de primaire applicatie of dienst waar je aan werkt: hoe lang duurt het over het algemeen om de dienst te herstellen nadat een wijziging in productie of een release naar gebruikers heeft geleid tot verminderde dienstverlening (bijvoorbeeld verstoring of uitval van de dienst) en vervolgens herstel vereist (bijvoorbeeld een hotfix, rollback, fix forward of patch)? Het herstellen houdt hier dus in
    - Net als een laag change fail percentage, is streven naar een lage failed deployment recovery time waardevol. Minder tijd besteden aan het rollbacken of fixen van brekende deployments. Doordat het team hier minder tijd aan kwijt is, kunnen ze meer tijd spenderen aan het leveren van nieuwe features of het kwalitatief beter maken van bestaande, aldus waarde voor de klant.

Bron: https://dora.dev/quickcheck/

Door ieder van deze metrieken in te vullen met een waarde die overeenkomt met hoe DORA het zelf invult (zie de quickcheck), is er een indicatie van hoe een team presteert in vergelijking met andere teams in de desbetreffende industrie. Voor iedere suggestie en oplossing die genoemd wordt in dit rapport zal er ook gezegd worden op welke metriek dit het meeste invloed heeft.

Het doel

Dit document is opgesteld om meer inzicht te geven in de huidige SDLC van Ace en hoe dit verbeterd kan worden. Dit zal gedaan worden op basis van DORA-metrics, zodat gemaakte veranderingen in de toekomst ook meetbare verbeteringen kunnen tonen. Deze inzichten zullen voor een concreter beeld zorgen, waarbij de teamlead en teamleden van Ace een onderbouwde roadmap krijgen, waarin te zien is welke stappen er gezet moeten worden richting verbetering. Dit alles zodat het team, efficiënter en kwalitatief beter werk kan leveren, zonder dat hiervoor werkplezier ingeleverd moet worden.

Het is ook goed om te weten dat dit document niet bedoelt als definitief plan maar vooral als leidraad waarmee Ace in gesprek kan over mogelijke verbeteringen.

Tijdens de gap sessie zijn de volgende doelen naar voren gekomen:

1. Evalueren van de huidige SDLC.
2. Verbeteringen in de huidige SDLC vinden plaats met de eisen van de klant in gedachten.

Huidige ontwikkelproces, wensen en uitdagingen

Om een beeld te krijgen van het huidige ontwikkelproces, de doelen die Ace graag wil bereiken en welke pijnpunten het ervaart of verwacht te ervaren, zijn er 2 Gap Analyse sessies uitgevoerd. De werkwijze is later beoordeeld in een DORA score.

Resultaten gap analyse

Het software-ontwikkelproces van Ace is in te delen in 7 fases: Plan, Refinement, Development, Testing, Acceptatie, Deployment en Monitoring. Onderstaande afbeelding is een weergave van het bord dat voortkwam uit de Gap Analyse sessies.

Hieronder staat een kort tekstueel overzicht van wat er bij iedere fase komt kijken, wat er verwacht wordt binnen deze fase en uitdagingen die momenteel spelen, of voort kunnen komen bij het werken naar een doel.

Plan

In de planfase worden er tickets opgesteld op basis van Initiatives en Epics. De doelgroep die in gedachten wordt gehouden, zijn externe ontwikkelaars (door middel van de API). Tickets worden opgedeeld in reguliere tickets en bugfixtickets.

Doelen	Uitdagingen
Tickets moeten een compleet beeld geven van al het werk dat er gedaan moet worden om een epic compleet af te ronden.	De tickets geven momenteel niet een volledig beeld van de uit te voeren taak.
Onderlinge verbinding tussen tickets en werk moet duidelijk zijn.	

Refinement

De Refinement heeft als doel om een duidelijk beeld te schetsen van het werk in de nabije toekomst. In een refinementsessie worden alle tickets besproken waarvan developen op het punt staat om te beginnen. Voorafgaand aan de sessie dient iedereen zich in te lezen in de inhoud van de tickets, zodat er tijdens de sessie geen tijd hoeft te worden verspild aan het introduceren van tickets.

Doelen	Uitdagingen
Het team moet een gezamenlijk beeld hebben van wat de tickets allemaal inhouden.	Momenteel is het niet altijd duidelijk welke tickets wel en/of niet “refined” zijn. Dit zorgt ervoor dat dat voorafgaand aan een refinementsessie nog uitgewerkt moet worden.
Tickets moeten duidelijk gelabeld zijn en als het een bugfix betreft, moet het duidelijk zijn vanuit welk ander ticket deze bug ontstaan is.	

Development

Ace maakt gebruik van een vergelijkbare strategie als Scaled Trunk Based Development, maar in tegenstelling tot die strategie maakt Ace geen gebruik van Release Branches. Releases doen ze door middel van tags, die soms op bugfix-branches worden aangemaakt.

Doelen	Uitdagingen
Ace wil graag het reviewen van hun code en het gebruik van Git volgens erkende succesvolle strategieën doen. Het aanmaken van bugfix-branches om releases vanaf te doen, willen ze indien mogelijk vervangen door een nettere aanpak.	Momenteel komen tickets en MR's van bepaalde onderdelen van de applicatie bij dezelfde mensen uit, wat kan zorgen voor zogenoemde "Knowledge Silos", waarbij 1 persoon als enige kennis bezit over een (groot) deel van de codebase. Wanneer deze persoon op vakantie gaat of vertrekt, kunnen er hierdoor problemen ontstaan.
Ace wil meer inzicht in wie veel invloed heeft op welke delen van de codebase, omdat momenteel veel vergelijkbare taken vaak bij dezelfde mensen terechtkomen.	Soms worden MR's door vrijwel iedereen binnen het team nagekeken. Dit kost veel tijd en is niet altijd nodig.

Testing

Testen binnen Ace wordt gedaan door de developers en een tester. Wanneer de developers de features af hebben gemaakt, en zelf hebben getest, wordt het naar de tester doorgestuurd. De tester test concreet alle features die in de eerstvolgende release meegenomen worden. Wanneer de tester alle testen succesvol heeft uitgevoerd, maakt hij een release tag aan op GitLab.

Doelen	Uitdagingen
Het Jiraticket moet de oorspronkelijke bron van het testen zijn; mogelijke testrapporten staan hierin opgenomen.	Het werk van de tester moet duidelijk en waardevol zijn. Er moet dus goed gekeken worden naar wat voor handmatige tests nog nodig zijn indien automatische tests geïmplementeerd worden.
Meer automatische tests zullen zorgen voor meer garantie en vertrouwen in de code. Ook zal dit de tijd die developers besteden aan het testen van hun code verminderen.	

Deployment

Deployen wordt gedaan door middel van de tool Rundeck, waarbij er een tag wordt aangemaakt op Gitlab en vervolgens wordt gedeployd via Rundeck.

De configuratie van Rundeck wordt bij Ace geleverd vanuit een ander team, waardoor zij er niet veel tijd aan kwijt zijn om het te onderhouden. Ace hecht hier waarde aan, omdat ze met 1 druk op de knop een specifieke tag kunnen deployen.

Doelen	Uitdagingen
Het deployen moet met één druk op de knop uitgevoerd kunnen (blijven) worden.	Er wordt momenteel geen gebruik gemaakt van deployments in GitLab of Release branches. Wat het overzicht kan verstoren.

Monitoring

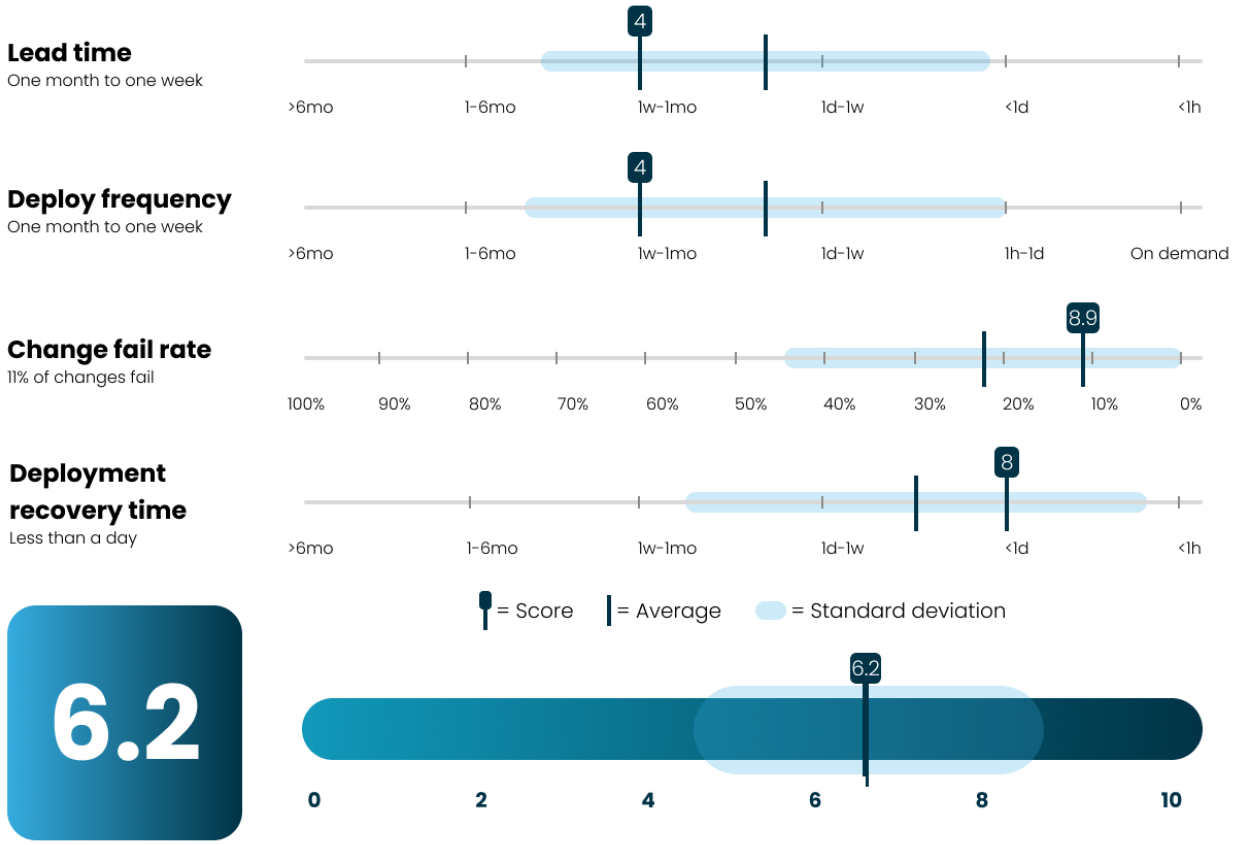
Tijdens de gehele SDLC speelt monitoring een cruciale rol. Dynatrace wordt ingezet voor diepgaande performance-analyse en probleemdetectie, terwijl Grafana wordt gebruikt om overzichtelijke dashboards te creëren voor realtime inzicht in hoe de software functioneert in productie.

Doelen	Uitdagingen
Monitoringoplossingen moeten kunnen draaien op het eigen cloudplatform.	De huidige alerts zijn, zowel in Dynatrace als Grafana, te complex.
Meegenomen log levels moeten gespecificeerd kunnen worden.	Binnen de dashboards wordt momenteel geen data weergegeven die wordt opgehaald uit Git.
Er moeten overzichtelijke GUI's zijn voor de in gebruik genomen dashboards.	Applicatielogging wordt niet doorgestuurd naar Dynatrace.

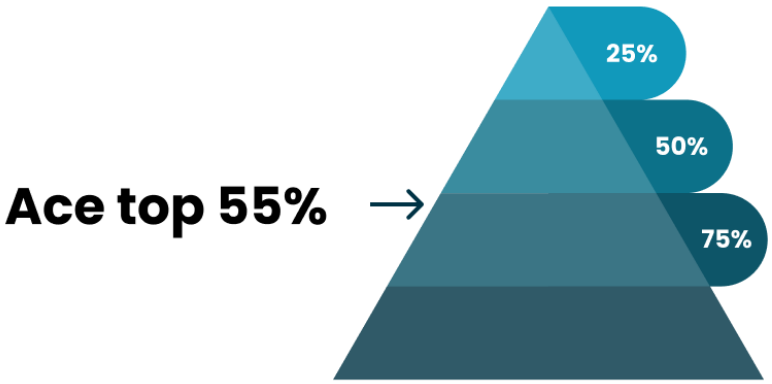
Huidige status DORA metrieken

Op basis van de kennis die Ace heeft gedeeld, is de volgende DORA-score gegeven aan hun huidige werkwijze. Dit is net ondergemiddeld en gezien de huidige processtructuur een verwacht resultaat. Belangrijk om te melden is dat er nog geen dashboards zijn ingericht om dit consistent inzichtelijk te krijgen, en de meegenomen waarden dus zijn gebaseerd op een momentopname.

De DORA-score van Ace wordt in dit rapport vergeleken met de gemiddelde scores van de gehele software-industrie, volgens de resultaten uit het 2023 DORA Rapport van Google. Bij iedere score is de standaarddeviatie ook weergegeven.


Lead time
One month to one week
4
>6mo
1-6mo
1w-1mo
1d-1w
<1d
<1h
Deploy frequency
One month to one week
4
>6mo
1-6mo
1w-1mo
1d-1w
1h-1d
On demand
Change fail rate
11% of changes fail
8.9
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Deployment recovery time
Less than a day
8
>6mo
1-6mo
1w-1mo
1d-1w
<1d
<1h

6.2
= Score
= Average
= Standard deviation
6.2
0
2
4
6
8
10


Ace top 55%
25%
50%
75%

Oplossingen

De huidige uitdagingen en wensen van Ace zijn gevat in 7 user stories. Voor deze user stories zijn verschillende oplossingen onderzocht, waarbij de keuze voor de oplossingen gebaseerd is op de wensen, uitdagingen en het huidige ontwikkelproces van Ace. Per user story worden de volgende oplossingen geadviseerd:

#	User story	Oplossingen
US-1	Als developer wil ik meer inzicht in het hele ontwikkelproces, zodat ik weet waar mijn werk aan bijdraagt.	Meer inzicht krijgen in Jira-tickets Dashboarding met Grafana
US-2	Als teamlead wil ik meer inzicht in de status van tickets en hoe lang het duurt deze te verwerken, zodat ik meer inzicht in het ontwikkelproces van mijn developers krijg en waar nodig bij kan sturen.	DORA-metrieken inzichtelijk maken met Apache Devlake Knowledge-silo's identificeren met CodeScene
US-3	Als teamlead wil ik dat merge requests sneller worden goedgekeurd, zodat het ontwikkelproces zo snel mogelijk wordt doorlopen, zonder dat hiervoor kwaliteit ingeleverd moet worden.	Pair programming Mob programming Mob testing (group review/testing)

#	User story	Oplossingen
US-4	Als developer wil ik waar mogelijk meer automatische tests in gebruik nemen die de veiligheid van onze software waarborgen, zodat ik minder tijd kwijt ben aan het handmatig controleren en testen hiervan en sterker kan onderbouwen dat we veilige software leveren.	Automatische Security Testing met GitLab GitLab Dependency scanning GitLab License scanning GitLab Secret detection
US-5	Als teamlead wil ik mijn releaseproces zo consistent en eenvoudig mogelijk maken voor mijn developers, zodat zij kunnen focussen op het ontwikkelen van de applicatie en niet beïnvloed worden door stappen buiten de controle van mijn developers.	Scaled Trunk Based Development • Feature Flags • Scaled Trunk Based Development GitLab deployments Canary Deployment
US-6	Als developer wil ik goede observability voor ons product, zodat ik met meer vertrouwen kan deployen naar productie, omdat problemen snel kunnen worden gevonden en hierdoor eerder gewerkt kan worden aan een oplossing.	Monitoring m.b.v. Dynatrace Dashboarding met Grafana

US-1

Als developer wil ik meer inzicht in het hele ontwikkelproces, zodat ik weet waar mijn werk aan bijdraagt.

Meer inzicht krijgen in Jira-tickets

Op dit moment ontbreekt er in het ontwikkelproces een duidelijk verband tussen de taken die de developers uitvoeren en de bredere doelstellingen van het team. Hierdoor is het lastig voor de developers om een helder overzicht te krijgen van hoe hun werk bijdraagt aan het grotere geheel. Dit zorgt er niet alleen voor dat de efficiëntie lager ligt, maar zorgt er ook voor dat het meten en verbeteren van belangrijke DORA-metrics, zoals lead time, change failure rate en deployment frequency, moeilijker wordt.

Tickets volledig uitwerken

Tickets bevatten momenteel vaak niet genoeg informatie of zijn bewust niet uitgewerkt. Dit leidt tot een gebrek aan kennis bij de developers over de volledige scope van een epic, iets wat kan leiden tot het vergeten van werkzaamheden en daardoor dus de lead time en cycle time negatief kan beïnvloeden.

Door de tickets wel allemaal volledig uit te werken, of in ieder geval allemaal aan te maken, ontstaat er een completer beeld van al het werk dat nodig is. Dit leidt tot:

- Door het minimaliseren van onduidelijkheden en het duidelijk hebben van welk werk er moet worden gedaan, zorgt ervoor dat er efficiënter gewerkt kan worden.
- Door alle tickets uit te werken kunnen afhankelijkheden en risico's al vooraf beter worden gedefinieerd, wat de kwaliteit van implementaties verbetert.

Labels toevoegen aan tickets

Om tickets nog beter uit te werken, wordt het gebruik van labels ook aangeraden. Op dit moment worden er geen labels gebruikt door Ace, waardoor er weinig tot geen inzicht in tickets is. Door

bijvoorbeeld de hieronder gegeven voorbeelden van labels toe te voegen, kan er veel meer inzicht in de Jira-tickets worden vergaard.

Knelpunten identificeren

Door labels te gebruiken die specifiek aangeven welke component(en) een ticket raakt, krijgen de developers een beter beeld van waar het meeste werk wordt gedaan en waar het meeste werk nog zit. Op basis hiervan kunnen knelpunten in het ontwikkelproces worden gevonden.

Door oplossingen voor deze knelpunten te vinden, kan het ontwikkelproces efficiënter worden gemaakt, waardoor er dus geld wordt bespaard.

Bugtracking

Het consistent labelen van bugs in tickets biedt duidelijke voordelen voor het inzichtelijk maken en het oplossen van problemen:

- Bugs kunnen direct worden gekoppeld aan gerelateerde features en componenten, wat het opsporen en oplossen ervan versimpelt.
- Door het gebruik van buglabels kunnen trends in foutmeldingen worden geïdentificeerd. Denk bijvoorbeeld aan een verhoogd aantal bugs in een component.
- Buglabels als `Critical` of `Minor` geven een urgentie aan bij een bug en kunnen daardoor helpen bij het plannen van bugfixes.

Door het inzichtelijk maken van bugtickets door bijvoorbeeld de bovenstaande voorbeelden, kunnen bugs veel efficiënter worden gevonden en opgelost.

Daarnaast gaat de kwaliteit van het eindproduct omhoog door het goed labelen van bugs. Door bugtickets te labelen met hun urgentie wordt er bijvoorbeeld voor gezorgd dat de grootste problemen in de applicatie sneller worden opgelost en de kwaliteit van het eindproduct groter is.

Dashboarding met Grafana

Op dit moment is er, naast het standaard zoeken naar tickets, geen effectieve manier om overzicht en inzicht te krijgen in de status van werkzaamheden, afhankelijkheden en knelpunten.

Door de onderstaande tools te gebruiken, kan dit overzicht wel worden gecreëerd, waardoor Ace hun proces kan verbeteren en kosten kan besparen.

Jira dashboards

Jira biedt dashboards aan waarmee teams overzichten kunnen creëren van tickets, op basis van bijvoorbeeld labels of de assignees. Deze dashboards zijn ideaal voor simpele use cases, zoals het bijhouden welke tickets nog openstaan, wie ergens aan werkt en voor het filteren op basis van zelfgemaakte labels.

Door snelle overzichten te creëren in Jira is er altijd snelle toegang tot de belangrijkste data en hoeft er geen tijd te worden besteed aan het handmatig zoeken naar tickets.

Grafana-dashboards

Een nadeel van Jira-dashboards is dat ze werken met vooraf gedefinieerde gadgets die slechts beperkte mogelijkheden voor aanpassing bieden. Voor complexere dashboards zijn er betaalde plugins nodig of kan er gebruik worden gemaakt van Grafana-dashboards.

Met behulp van de Jira-plugin voor Grafana kan er in tegenstelling tot de gelimiteerde gadgets van Jira met veel meer vrijheid data worden gevisualiseerd.

https://grafana.com/grafana/plugins/grafana-jira-datasource/

US-2

Als teamlead wil ik meer inzicht in de status van tickets en hoe lang het duurt deze te verwerken, zodat ik meer inzicht in het ontwikkelproces van mijn developers krijg en waar nodig bij kan sturen.

DORA-metrics zijn belangrijk voor teams die willen groeien in kwaliteit en efficiëntie. Ze bieden een duidelijke en meetbare manier om te zien hoe goed een team presteert. Als teams geen DORA-metrics meten, kan dat nadelige gevolgen hebben:

- **Geen inzicht in prestaties:**
  Zonder DORA-metrics is het lastig om te weten hoe goed een team werkt. Problemen zoals lange doorlooptijden of veel fouten bij implementaties worden vaak niet opgemerkt, waardoor ze blijven bestaan.
- **Geen duidelijke verbeterdoelen:**
  DORA-metrics laten zien wat goed gaat en wat beter kan. Zonder deze informatie wordt het moeilijker om te bepalen waar het team zich op moet richten, met het risico dat er op de verkeerde dingen wordt gefocust.

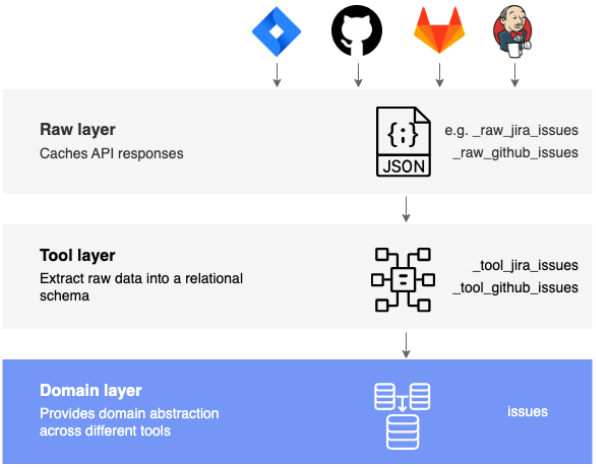
DORA-metrieken inzichtelijk maken met Apache Devlake

Om DORA-metrics te meten kan ervoor worden gekozen om handmatig door alle data heen te gaan om de benodigde informatie te verzamelen, maar er zijn tegenwoordig ook tools voor beschikbaar. De tool die wij aanbevelen is Apache DevLake.

Apache DevLake is een opensourceproduct dat data uit DevOps-tools inzichtelijk maakt om bijvoorbeeld DORA-metrics te genereren. Naast DORA-metrics geeft het ook inzicht in andere metrics die specifiek voor engineeringleads zijn bedoeld.

https://devlake.apache.org/

Apache DevLake werkt door met een van de vele sources zoals Jira en GitLab te praten en daaruit alle benodigde data te halen. Deze data wordt vervolgens omgezet naar een domein dat gebruikt kan worden om dashboards mee te maken.



Bron: https://devlake.apache.org/docs/Overview/Architecture

Voor dit domein zijn al een aantal ingebouwde dashboards aanwezig die bijvoorbeeld de DORA-metrics weergeven, zodat gebruikers relatief makkelijk inzicht kunnen krijgen in hun ontwikkelproces. Wanneer de gebruiker inzichten mist in de standaarddashboards, is er ook de mogelijkheid om met het domein eigen dashboards te ontwerpen.

Het gebruik van Apache DevLake heeft meerdere voordelen:

- Geautomatiseerde verzameling en berekening van DORA-metrics vanuit Jira en GitLab.
- Ingebouwde dashboards, waardoor er zonder extra configuratie een goed beeld van de prestaties van een team ontstaat.
- Naadloze integratie in Grafana, waardoor de DORA-metrics binnen al bestaande omgevingen kunnen worden bekeken.

Door het gebruik van Apache DevLake kan er enorm veel tijd worden bespaart, doordat er geen tijd besteed hoeft te worden aan het berekenen van DORA-metrics. Daarnaast zorgen deze DORA-metrics weer voor duidelijke verbeterdoelen door het tonen van meetbare resultaten over het ontwikkelproces van het team.

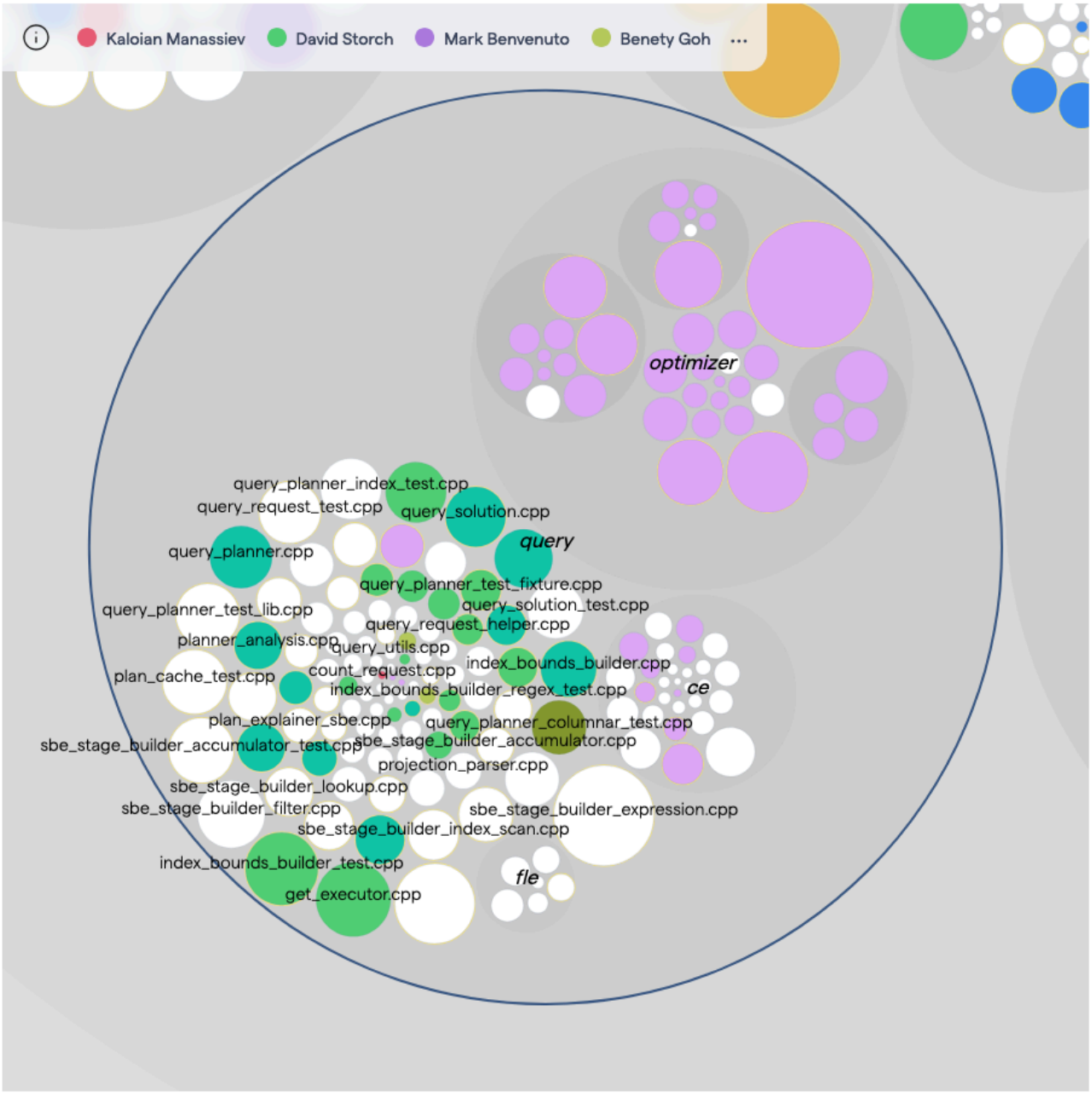
Knowledge-silo's identificeren met CodeScene

Knowledge-silo's, ook wel knowledge-islands genoemd, ontstaan wanneer het merendeel van een deel van de codebase wordt geschreven door één developer. Dit is een risico voor het developmentteam, omdat de kans bestaat dat deze developer uitvalt en er vervolgens te weinig kennis is binnen de rest van het team om verder te kunnen werken aan de relevante onderdelen.

Men noemt dit concept ook weleens de “Bus Factor”. Dit houdt in principe in dat men zegt: “Hoeveel developers moeten door een bus aangereden worden voordat de kennis verloren is van

dit stuk code?" Hoe hoger de Bus Factor, hoe minder het onderdeel in kwestie een Knowledge Silo is.

Op dit moment is er geen inzicht in deze knowledge silo's en is het dus niet duidelijk of Ace het hierboven beschreven risico loopt. Om hier inzicht in te krijgen, kunnen er tools worden gebruikt die de Knowledge Silo's visualiseren. Dit is nog niet een veelgebruikt iets, dus de opties zijn gelimiteerd. De bekendste hiervan is CodeScene. Alternatieve tools zijn er momenteel nog niet veel; echter, men kan op basis van een repository en wat Python-scripts de silo's toch nog visualiseren. Een voorbeeld van hoe CodeScene dit weergeeft, staat hieronder:



1 Bron: https://codescene.io/projects/32159/jobs/2792745/results/social/knowledge/individuals

CodeScene kan zelf gehost worden, maar hier is geen open-source- en/of gratis variant van. Ons advies hierin is om het eenmalig te installeren en te koppelen met GitLab. Dit kan met een gratis trial. Door dit te doen, kan er gratis een beeld gekregen worden van mogelijke knowledge silo's die momenteel bestaan.

Door Knowledge Silo's te identificeren, kan Ace m.b.v. bijvoorbeeld pair programming en mob programming ervoor zorgen dat de bus factor wordt verhoogd en de Knowledge Silo's dus verdwijnen. Dit zorgt ervoor dat er zonder problemen mensen uit het team kunnen vertrekken en dat het schrijven van code en reviewen van bepaalde MR's niet afhankelijk is van één enkel persoon. Dit zal de lead time binnen Ace verlagen, doordat er meer mensen zijn die genoeg kennis hebben om code te kunnen schrijven en reviewen.

Maar wat belangrijk is, is dat de silo's niet visueel zichtbaar moeten zijn om ze te erkennen en er wat aan te doen. Er kan door middel van kennisdeling altijd ook voor gezorgd worden dat je de kans vermindert dat je in de toekomst nog Knowledge Silo's creëert, wat in het voordeel van het developmentteam is. Aan het verminderen draagt daarom US-3 ook bij.

US-3

Als teamlead wil ik dat merge requests sneller worden goedgekeurd, zodat het ontwikkelproces zo snel mogelijk wordt doorlopen, zonder dat hiervoor kwaliteit ingeleverd moet worden.

Pair programming

De basis van mob-programming en -testing ligt in pair-programming. Binnen pair programming is er veel variatie in de uitvoering van het werk. De volgende waarheden zijn echter standaard aanwezig:

- Er zijn twee programmeurs samen aan het ontwikkelen van design, realisatie tot en met testing
- Een van beide programmeurs typt op een moment van tijd en de ander navigeert.
- Het doel is om te leren van de andere deelnemer over de codebase en in het uitvoeren van het werk te leren van de aanpak van de ander.
- Het inzetten van pair programming wordt gedaan bij gemiddelde en hoge complexiteitsproblemen.
- Er is een circa 15% efficiëntieverlies van het losgekoppeld laten ontwikkelen van programmeurs.

Door pair programming toe te passen kunnen gemakkelijker gaten in de kennis van ontwikkelaars over de codebase worden gedicht zonder veel efficiëntie te verliezen. Op de langere termijn zal dit de noodzaak van met meerdere mensen reviewen verminderen en de knowledge silo's weten te verminderen die gevonden zijn.

Wat ondersteunende informatie om dit toe te passen:

https://medium.com/@maaretp/the-driver-navigator-in-strong-style-pairing-2df0ecb4f657
https://raygun.com/blog/how-good-is-pair-programming-really/

Mob programming

Mob programming is een geavanceerde vorm van pair programming. Bij pair programming is er één iemand die achter het toetsenbord zit en één iemand die nadenkt over wat er getypt moet worden. Bij mob programming komt hier een extra rol bij kijken. Mob programming kent de volgende rollen:

- Driver
- Navigator (of coördinator)
- Mob

De **Driver** is, net als bij pair programming, degene die het daadwerkelijke typewerk op zich neemt. Dit wordt gedaan op basis van instructies die door de **Navigator** worden gegeven. De navigator formuleert deze instructies op basis van wat de **Mob** allemaal suggereert. In dit geval is de mob de overige developers aanwezig, die onderling zitten te sparren over wat er geïmplementeerd moet worden.

Mob programming wordt gebruikt bij complexe problemen om zo sneller een oplossing te vinden voor het probleem. Daarnaast zal het, net als pair programming, de knowledge silos weten te verminderen door het delen van kennis tijdens de sessies. Ook zal de kwaliteit van de code omhooggaan doordat de code samen wordt geschreven, iets wat vervolgens ook zal resulteren in kortere reviewtijd en minder benodigde reviews. Omdat er al genoeg developers naar hebben gekeken bij het schrijven van de code, zullen er minder naar hoeven kijken achteraf bij de review.

Zie de bron hieronder voor extra informatie hierover:

https://medium.com/@alexdh359/boosting-team-performance-with-mob-programming-a-guide-for-scrum-masters-fa758bc60172

Mob testing

Group reviews (of Mob review), wat hand-in-hand gaat met Mob testing, is een aanpak waarbij 3 of meer developers samen een feature gaan reviewen/testen. Hiervoor zijn verschillende strategieën mogelijk. Maar in de basis is het vaak hetzelfde. Men doet samen een stel exploratietests, waarbij iedereens input wordt gebruikt om te kijken wat er gedaan wordt met de software. Mob-testing kent in dit geval twee rollen:

- Driver
- Navigator

De **Driver** is, net als bij mob programming, degene die het typewerk doet. Bij mob testing is het echter gebruikelijk dat deze rol binnen de group rouleert.

De **Navigators** zijn in dit geval meerdere mensen. Iedereen heeft inspraak in wat er allemaal getest wordt en hoe dit gedaan wordt. De navigators zijn verantwoordelijk voor het geven van instructies aan de driver en het bedenken, opstellen en doorlopen van testcases.

Door het uitvoeren van mob testing kunnen problemen sneller worden geïdentificeerd, waardoor er minder tijd zit in het zoeken naar de oorzaak van een probleem en meer gespendeerd kan worden aan het oplossen. Hierdoor zal de efficiëntie van Ace en de kwaliteit van het geleverde werk omhooggaan. Dit zal uiteindelijk dus gaan resulteren in een lagere lead time en in sommige gevallen zelfs een lagere failed deployment recovery time.

Het onderstaande artikel behandelt mob testing verder:

https://www.testlearning.net/en/posts/mob-testing

US-4

Als developer wil ik waar mogelijk meer automatische tests in gebruik nemen die de veiligheid van onze software waarborgen, zodat ik minder tijd kwijt ben aan het handmatig controleren en testen hiervan en sterker kan onderbouwen dat we veilige software leveren.

Automatische Security Testing met GitLab

Change Fail Percentage —

Hieronder staan verschillende oplossingen die geïmplementeerd kunnen worden voor het uitvoeren van automatische securitytests. Met tot slot een onderbouwing voor waarom wij vinden dat GitLab SAST en DAST de beste optie zijn voor Ace.

GitLab SAST & DAST

SAST is een vorm van testen die in GitLab-pipelines aangezet kan worden, waarbij men hun code automatisch kan laten scannen op securitygebied door een tool die de sourcecode analyseert. Hierbij wordt er gekeken naar of er onveilige stukken code of traces gevonden kunnen worden. SAST beschikt over de mogelijkheid om verschillende talen en frameworks te ondersteunen. Hiervoor maakt het gebruik van een aantal zogeheten “analyzers”. Een overzicht van ondersteunde frameworks en bijhorende analyzers is te vinden in de GitLab-docs:

https://docs.gitlab.com/ee/user/application_security/sast/#supported-languages-and-frameworks

DAST is net als SAST een vorm van automatisch testen op securitygebied. Waar SAST kijkt naar de code en logica hiervan om te zien of er een securityprobleem is, gaat DAST van buiten de code af, in een testvariant van de app, verzoeken sturen naar de software om te kijken of er onverwachte resultaten terugkomen. Denk hierbij aan het afvuren van allerlei verschillende verzoeken naar een API die met de software communiceert.

De volledige functionaliteit van DAST is enkel beschikbaar in GitLab Ultimate. In GitLab Free en Premium kan er wel gebruik gemaakt worden van de CI/CD componenten van dependency scanning. Hierdoor kunnen de automatische tests wel uitgevoerd worden, alleen worden de resultaten niet volledig opgenomen in het security report binnen de GitLab UI.

De GitLab-docs behandelen verder hoe DAST precies werkt en gebruikt kan worden:

https://docs.gitlab.com/ee/user/application_security/dast/

Dynatrace IAST

IAST is een combinatie van SAST en DAST. IAST werkt alleen met talen die een virtuele runtime-omgeving hebben, zoals Java, C#, Python en Node.js. Dynatrace is wel een tool die pas werkt zodra software gedeployd is. Dit betekent dat IAST pas toegepast kan worden zodra de software draait in een acceptatie-, test- of productieomgeving.

Sonar deeper SAST

Sonar bezit ook over SAST om zo het risico op security breaches te verlagen. Dit doen ze door de volledige codebase te scannen en analyseren op kwetsbaarheden, bugs en codesmells om zo de kwaliteit van de code en de security te kunnen garanderen.

Advies

Omdat GitLab zowel SAST als DAST aanbiedt en dit binnen een pipeline gedraaid wordt voordat de code gedeployd is naar een acceptatie- of testomgeving, adviseren wij dit als beste oplossing. Sonar Deeper SAST is enkel code-analyse en mist het dynamische blackboxtesten van DAST. Hoewel Dynatrace IAST juist wel het dynamische blackboxtesten bevat, zijn wij van mening dat het uitvoeren van securitytests wanneer de software al gedeployd is eigenlijk te laat is. Idealiter wil je dit doen voor deployment. Daarom adviseren wij om dit binnen GitLab-pipelines te doen.

GitLab dependency Scanning

Dependency scanning analyseert de dependencies van je applicatie voor bekende vulnerabilities. Hierbij worden ook de geneste dependencies meegenomen.

Dependency scanning kan in de pipeline worden gedraaid en laat in het mergerequest zien welke vulnerabilities met welke severity er zijn gevonden.

Daarnaast kan het ook buiten pipelines om worden uitgevoerd door continuous vulnerability scanning.

De volledige functionaliteit van Dependency Scanning is, net als DAST, enkel beschikbaar in GitLab Ultimate. Wederom kan er wel gebruik gemaakt worden van de benodigde componenten om de automatische tests uit te voeren. In dit geval is het ook weer zo dat de resultaten enkel niet volledig verwerkt worden binnen de UI.

https://docs.gitlab.com/ee/user/application_security/dependency_scanning/

GitLab license Scanning

License scanning van CycloneDX-files kan worden gebruikt om de, door de dependency scanning CI-jobs, gegenereerde CycloneDX software bill of materials (SBOM) te analyseren. Deze methode kan 600 verschillende typen licenses identificeren. Hiernaast sluit dit goed aan bij regelgeving

https://docs.gitlab.com/ee/user/compliance/license_scanning_of_cyclonedx_files/

GitLab secret Detection

GitLab Secret detection

De secret detection van GitLab wordt gebruikt om private keys, tokens en wachtwoorden te vinden in de code. Er zijn meerdere lagen aan security om ervoor te zorgen dat deze 'secrets' niet per ongeluk in de repository terechtkomen:

- **Secret push protection**: wanneer je pusht naar GitLab worden de commits gecontroleerd op secrets. Deze push wordt geblokkeerd wanneer er secrets worden gevonden, tenzij de gebruiker de secret detection skipt.
- **Pipeline secret detection:** wanneer je secret detection toevoegt aan de CI/CD kun je bijvoorbeeld scannen op secrets wanneer je een merge request naar de main branch opent, zodat er nooit op deze branch terechtkomen.
- **Client-side secret detection:** deze laag zorgt ervoor dat er niet per ongeluk secrets in comments of descriptions van merge requests terecht kunnen komen. Voordat de secrets namelijk naar GitLab worden gesaved, krijgt de gebruiker de optie om de input aan te passen.

https://docs.gitlab.com/ee/user/application_security/secret_detection/

Sonar secret detection

Sonar beschikt ook over een vorm van Secret Detection. Deze wordt automatisch uitgevoerd en zien wij als een sanity check.

https://www.sonarsource.com/solutions/secrets-detection/

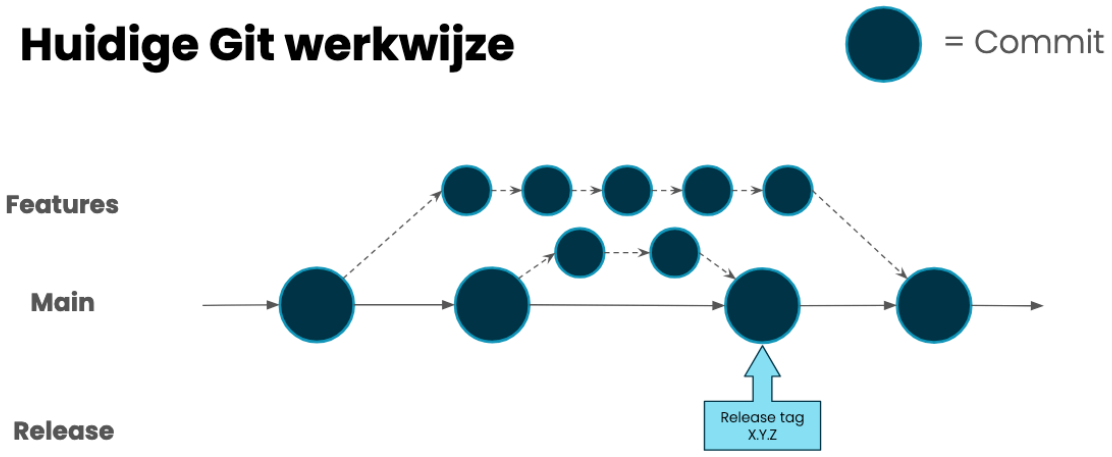
US-5

Als teamlead wil ik mijn releaseproces zo consistent en eenvoudig mogelijk maken voor mijn developers, zodat zij kunnen focussen op het ontwikkelen van de applicatie en niet beïnvloed worden door stappen buiten de controle van mijn developers.

(Scaled) Trunk Based Development

Lead Time +

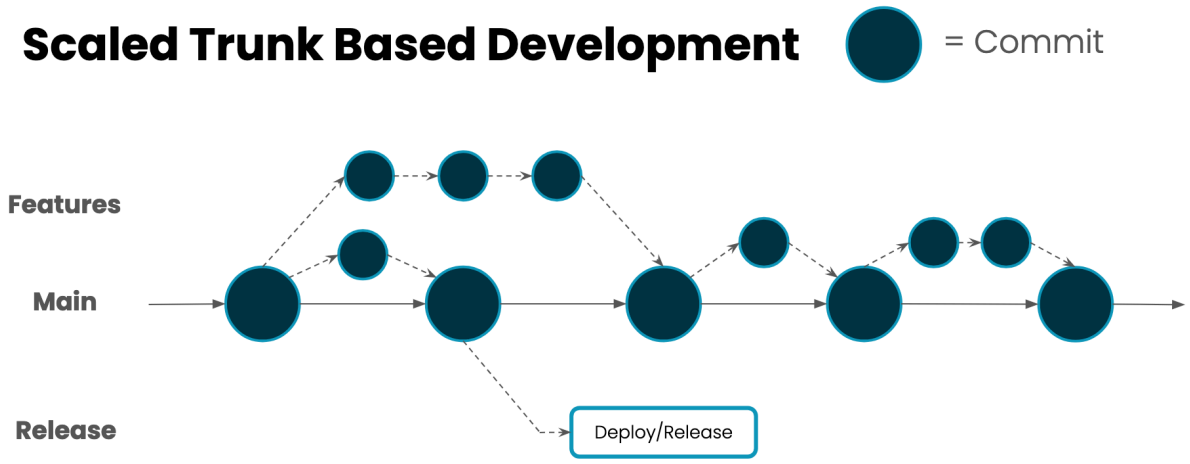
Ace gebruikt al een git-strategie die lijkt op Scaled Trunk Based Development, maar hier nog een beetje van afwijkt. De afbeelding hieronder schetst hoe wij hebben begrepen dat Ace momenteel werkt:



AVISI

Echter, wij zien waarde in het gebruiken van Scaled Trunk Based Development. Scaled Trunk Based Development verschilt van Trunk Based Development in dat het wel gebruikmaakt van feature branches. Voor de rest volgen de principes van Trunk Based Developments. De feature branches zijn short-lived, testomgevingen worden opgezet vanaf commits naar de main branch

en releases worden gedaan vanuit release branches. De afbeelding hieronder schetst hoe Scaled Trunk Based Development werkt:



De voornaamste reden dat wij dit aanraden is de verhoging van de development snelheid. En de positieve impact die het zal hebben op de kwaliteit van de geleverde software. Momenteel worden releases gedaan op basis van een tag. Bij Trunk Based Development kan dit nog steeds, maar dan vanuit releasebranches. Release branches bieden meer mogelijkheden op het gebied van releases en deployment vanuit GitLab. Bijvoorbeeld het automatisch deployen naar een acceptatieomgeving, zodra de releasebranch is aangemaakt. Ook kan er makkelijk voor gezorgd worden dat elk MR dat naar main wordt gemerged automatisch een testomgeving opstart, zodat de tester hier meteen mee aan het werk gaat.

De volgende onderdelen moeten fatsoenlijk geïmplementeerd worden voordat Ace over kan schakelen naar volwaardig Trunk Based developen:

- Meerdere testomgevingen
- Feature flags

Ace bezit al over de mogelijkheid om lokaal verschillende testinstanties op te zetten van zijn software. Hierdoor kan de tester vooruit/parallel werken aan het ontwikkelteam. Echter wordt er standaard getest op de testomgeving, die gebouwd wordt op basis van de code zoals die op de default branch staat. Soms is de tester nog bezig met het testen op de testinstantie en zorgt dit ervoor dat developers niet hun code kunnen mergen naar de default branch. Hierom is het daadwerkelijk opzetten van meerdere testomgevingen, op basis van elke merge naar main, een belangrijk onderdeel van echt (Scaled) Trunk Based developen. Elke instantie staat dan klaar

voor de tester om te werk te gaan, zodra de tester hier klaar mee is, wordt deze weggegooid. Zo wordt er altijd getest op basis van hoe de nieuwe feature werkt op de huidige status van de default branch. Op deze manier zitten developers ook nooit te wachten tot ze van de tester te horen krijgen dat er weer merge requests naar de default branch doorgevoerd kunnen worden.

Feature flags worden later in dit hoofdstuk behandeld.

Nadat Trunk Based Development in gebruik is genomen, kan er ook makkelijk overgeschakeld worden naar het gebruik van Canary Deployments. Wat ook later in dit hoofdstuk behandeld wordt.

GitLab Deployment

Momenteel wordt de tool Rundeck gebruikt voor het deployen van een release. Dit werkt voor Ace goed. Wanneer er gebruikgemaakt wordt van releasebranches (volgens de principes van Scaled Trunk Based Development), wordt de waarde van het Rundeck steeds minder. De stappen die in Rundeck doorlopen worden, kunnen op dat punt makkelijk gedaan worden binnen een GitLab-pipeline die enkel afgevuurd wordt op release-branches, vervolgens de tag automatisch aanmaakt en deze koppelt aan de release/deployment. Dit neemt een extra tool weg uit het landschap en geeft daarmee extra verantwoordelijkheid aan Ace. Doordat Ace verantwoordelijk is voor hun deployment, kunnen zij dit ook verder zelf inrichten op een manier die hun deployment ondersteunt en eventueel versimpelt..

Feature Flags

Feature flags bieden Ace de optie om flexibel te kunnen bepalen wat wel en niet momenteel in productie draait. Bij Trunk Based Development zijn feature flags een krachtige manier om te kunnen zorgen dat een feature die al op productie staat uitgezet kan worden.

Doordat features live-gezet kunnen worden terwijl ze nog niet in gebruik genomen zijn, betekent het dat er continu gedeployed kan worden en minimale aanpassingen nodig zijn voor het activeren van features. Wat belangrijk is bij het gebruik van feature flags, is dat wanneer een flag niet meer nodig is, deze ook verwijderd wordt. Wij adviseren daarom om bij elke release een overzicht bij te houden van bestaande en nieuw geïntroduceerde feature flags, zodat deze op een vast moment (de release van een nieuwe versie) nagelopen kunnen worden.

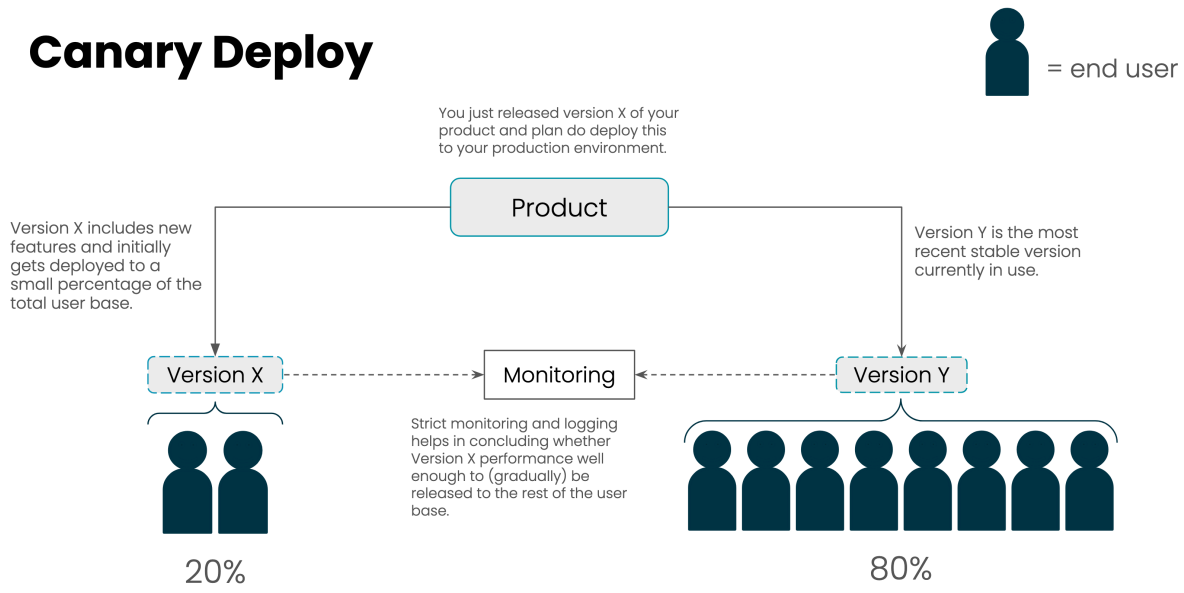
Er moet wel een besluit genomen worden over hoe Ace feature flags wil gaan implementeren. Ace heeft in het verleden al feature flags gebruikt en een deel hiervan hebben ze nog in hun codebase. Echter, het is belangrijk om gestructureerd te werk te gaan met feature flags. Het advies aan Ace is daarom om nogmaals te kijken naar hun gebruik van feature flags en in te zien hoe feature flags in combinatie met strenge Scaled Trunk Based Development en gescheiden test-/acceptatieomgevingen meerwaarde kan bieden voor hun SDLC.

Het is bekend dat feature flags lastig te combineren zijn met databasemigraties. Ace maakt al gebruik van een strategie om dit gestructureerd te doen, belangrijk blijft dat hier rekening mee gehouden blijft worden zodat doorgevoerde aanpassingen altijd nog teruggedraaid kunnen worden.

Canary Deployment

Change Fail Percentage –

Canary Deployment is een deploystrategie die gebruikt wordt om een nieuwe versie van software in productie een tijdje in een publieke test te laten draaien. Zodra er een nieuwe versie van het product beschikbaar is, wordt deze gedeployd en krijgt een klein percentage van de eindgebruikers toegang tot deze versie, in plaats van de huidige stabiele versie. Initieel zal er gekeken worden of de nieuwe versie stabiel is en er geen problemen worden ervaren door de gebruikers. Zodra dit blijkt, wordt er langzaam incrementeel overgeschakeld naar de nieuwe versie. Van 20% gaat het naar bijvoorbeeld 40%, en indien er bevonden wordt dat deze goed presteert, wordt er langzaam gewerkt naar 100%. Zie de afbeelding hieronder voor een beeld van het idee achter Canary Deploys.



Hierbij kunnen versie X en versie Y aparte versies zijn van de API die Ace aanbiedt. Belangrijk bij Canary Deployments is goede monitoring. Zonder goede monitoring is het niet met zekerheid te zeggen dat de Canary Deploy (versie X in de afbeelding) goed genoeg presteert om er de gehele userbase hiernaartoe om te schakelen. Of de prestatie goed genoeg is voor de gehele userbase is afhankelijk van de non-functional requirements die afgesproken zijn. Denk hierbij aan security, response time, usability, etc. Door dit allemaal te monitoren, krijgt het team inzicht in of de nieuwe versie bevalt bij de userbase. Idealiter focus je hierbij dan ook op meetbare parameters die je actief kan monitoren, bijvoorbeeld met behulp van Dynatrace.

Canary Deployments werken hand in hand met het opzetten van nieuwe testomgevingen en feature flags. In het voorbeeld hierboven kunnen bij versie X een stel feature flags op 'actief' gezet worden die bij versie Y nog niet actief zijn,. Zodradeze features als bruikbaar worden ervaren, kunnen de flags bij versie Y ook op actief gezet worden, zonder dat hiervoor een nieuwe versie gebouwd en gedeployd moet worden.

Vooral sterk voor Ace is dat er zonder rollback of andere ingreep een feature uitgezet kan worden met een feature flag, of in theorie een “oudere versie” kan worden aangezet voor de gehele gebruikersgroep. Door Canary Deployments als default-strategie in te regelen voor het deployen van nieuwe versies en het aanzetten van nieuwe features, wordt het release- en acceptatieproces meer gestroomlijnd.

US-6

Als developer wil ik goede observability voor ons product, zodat ik met meer vertrouwen kan deployen naar productie, omdat problemen snel kunnen worden gevonden en hierdoor eerder gewerkt kan worden aan een oplossing.

Monitoring m.b.v. Dynatrace

Aangezien Ace Dynatrace al gebruikt, ligt het voor de hand om de monitoring van de applicatie hierdoor te laten regelen. Dynatrace is een alles-in-één-oplossing voor monitoring. Met behulp van de Davis AI-engine wordt er continu data geanalyseerd om contextspecifieke inzichten te bieden.

Dynatrace heeft automatische ontdekking, waardoor het platform nieuwe hosts, VM's en containers automatisch kan ontdekken en monitoren.

Dynatrace is een complexe tool, omdat deze zo uitgebreid is, waardoor het voor nieuwe gebruikers een tijdje duurt voordat ze het helemaal onder de knie hebben. Daarnaast is er het gevaar dat de developers door het gebruik van Dynatrace te veel gaan leunen op AI, waardoor er expertise op het gebied van logging verloren gaat.

Aangezien Avisi een investering gaat doen in Dynatrace, adviseren wij Ace om Dynatrace in gebruik te nemen en te gaan kijken of het voor hen een goede oplossing is.

Dashboarding met Grafana

Naast Dynatrace adviseren we ook om Grafana te blijven gebruiken voor het bouwen van dashboards. De dashboards kunnen namelijk gebruikt worden door Apache DevLake voor de

DORA-metrics en om informatie over de productieomgeving te tonen op schermen in de kantoorruimtes.

Roadmap

De roadmap is gebaseerd op de prioriteitenlijst van Ace, de belangrijkste verbeterpunten tijdens de gap-sessie en de volgordelijkheid van realisatie. Rekening houdend met het voorgaande, stellen wij de volgende gefaseerde planning voor om de genoemde oplossingen te realiseren:

Fase	Toelichting	User stories	Oplossingen	Informatie	Tijdsbesteding
Fase 1 Meten & Werkwijze **Totaal tijdschatting:** 140 uur **DORA metrics estimation:** • Lagere failure rate • marginale verbetering lead time • inzicht in metrics en de realiteit	Fase 1 focust op het verkrijgen van een beter beeld van hoe er momenteel gepresteerd wordt, door het creëren van meer inzicht. Dit wordt gedaan op basis van dashboards voor Jira-tickets en het verzamelen van DORA- en Merge Request-metrieken. Daarnaast adviseren wij om wat laaghangend fruit op het gebied van developen en testen alvast op te maken, om wat vroege verbetering in te voeren.	US-2	DORA-metrieken inzichtelijk maken met Apache Devlake	Vanaf het begin meten zorgt ervoor dat het doorvoeren van oplossingen in impact kan worden gemeten en ook of de gemaakte aannames binnen deze rapportage valide zijn.	32 uur

Fase	Toelichting	User stories	Oplossingen	Informatie	Tijdsbesteding
DORA: 6.6		US-1	Meer inzicht in tickets door Jira-labels en Grafana-dashboards	Tickets goed labelen, zodat het duidelijk is waar het op doelt en of deze refined zijn of niet, is stap 1. Vervolgens kan er nog gekeken worden of hier al iets extra's aan dashboarding toegevoegd kan worden.	Te doen nadat er eigenaarschap is over dashboarding van Grafana 8 uur
		US-3	Pair programming standaardiseren	In de moeilijkere stukken van de codebase waar met 5 man wordt gereviewd een pair programming aanpak toepassen om het aantal benodigde ogen langzaam te kunnen verminderen door een hogere mate van begrip te creëren in de codebase.	2 uur * aantal developers = 12 uur aan training

Fase	Toelichting	User stories	Oplossingen	Informatie	Tijdsbesteding
		US-3	Mob-programming en mob-testing sessie inplannen.	Plannen van een initiële mob-programming- en mob-testing-sessie om te kijken hoe dit bevalt	6 x 2 uur (mob programming) 6 x 1 uur (mob testing)
		US-5	(Scaled) Trunk Based Development/Release branches gebruiken	De eerste stap is gekoppeld aan het opzetten van meer testomgevingen en het gebruikmaken van releasebranches. Deze tijdsinvestering betreft alleen releasebranches	16 uur
		US-5	Testomgevingen parallel opzetten	Het opzetten van meer testomgevingen, zodat meer features tegelijk getest/naar acceptatie kunnen. Zonder dat deze omgevingen een blocker zijn voor nieuwe commits.	40 uur

Fase	Toelichting	User stories	Oplossingen	Informatie	Tijdsbesteding
		US-4	Automatische tests in GitLab toevoegen.	Het toevoegen van automatische tests binnen de GitLab pipelines is weinig werk en biedt al snel waarde.	4 + 2 uur * project

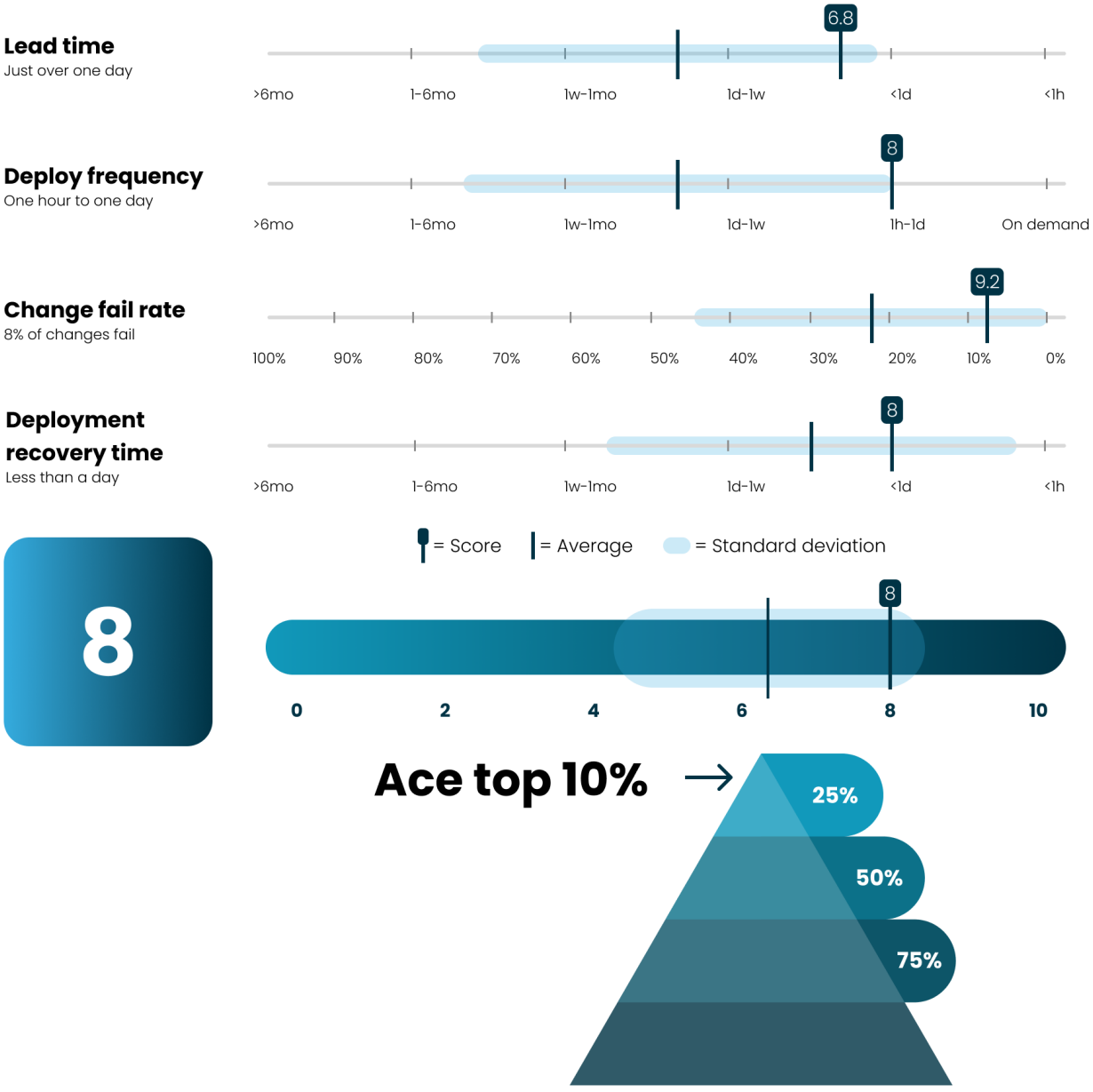
Fase	Toelichting	User stories	Oplossingen	Informatie	Tijdsbesteding
Fase 2 Development & Releases **Totale tijdschatting:** 128 uur **DORA metrics estimation:** • kleine toename in failure rate • grote verbetering lead time • veel snellere deploy frequency	Zodra er meer inzicht is en er wat eenvoudige eerste stappen zijn gezet, willen we focussen op het tackelen van de grotere pijnpunten. Hierbij zal de werkwijze van Ace sterk veranderd moeten worden, doordat vooral de gehele deployment- en releasestrategie moet worden veranderd.	US-5	Feature flags	Feature flags zijn noodzakelijk om de snelheid die gewonnen wordt bekwaam te benutten. Het controle nemen over de releasecycle dwingt af dat er processen en technologie is die de lead time zo min mogelijk afremt. De tijdsbesteding is zo ingestoken vanwege de noodzaak tot het doen van onderzoek om de juiste libraries en methodieken te vinden om goed feature flags in te richten binnen de applicatie.	60-80 uur

Fase	Toelichting	User stories	Oplossingen	Informatie	Tijdsbesteding
• grote verbetering in mean time to restore DORA-score: tussen 6,6 en 7.8, gebaseerd op bekwaamheid		US-5	Deployment/Releases vanuit GitLab	Het omzetten van Rundeck naar een Flux/release pipeline in GitLab heeft veel impact door de hoeveelheid werk die Rundeck op zich neemt.	16-48 uur

Fase	Toelichting	User stories	Oplossingen	Informatie	Tijdsbesteding
Fase 3 Deployment & Monitoring **Totale tijdschatting:** 84 uur **DORA metrics estimation:** • Grote vooruitgang in mean time to restore • Kleine verbetering in lead time	In de laatste fase zal er gefocust worden op de live-omgeving. Meer inzicht in de werking van de applicatie en nog meer zekerheid in de werking van de applicatie via canary deployments. Hierdoor is het ook gemakkelijker om grotere features en fouten direct terug te zetten en te valideren of daadwerkelijk het probleem is verholpen live.	US-6	Beter gebruikmaken van monitoring	Het inregelen van betere monitoring, wat nodig is bij het doorvoeren van veranderingen op het gebied van deployment.	Erg afhankelijk van wat er nu beschikbaar is en hoe snel nu fouten worden gevonden 6-24 uur

Fase	Toelichting	User stories	Oplossingen	Informatie	Tijdsbesteding
• Redelijke verbetering in failure rate DORA-score: tussen 7,4 en 8,0		US-5	Canary Deployment gebruiken	Door proces en technologie in te regelen op canary deployments is het mogelijk om meerdere versies (in samenwerking met feature flags) live te zetten.	40-60 uur

Naar onze schatting zal Ace na het uitvoeren van alle fasen uitkomen met de volgende DORA score:



Kostenoverzicht

Licenties

Voor de momenteel gesuggereerde oplossingen, zullen er geen extra kosten gemaakt hoeven worden voor het verkrijgen van nieuwe licenties. Dit komt doordat er gefocust is op het gebruiken

van open source en gratis tooling, of tooling die door middel van een gratis trial gebruikt kan worden voor het doeleinde van de oplossing.

Waar mogelijk wel extra kosten aan hangen, zijn de resources die gebruikt worden voor het draaien van pipelines. Het introduceren van de geadviseerde oplossingen zal leiden tot langere pipelines in GitLab, waar indirect kosten aan verbonden zijn. Hoeveel dit exact is, kan op dit moment nog niet gezegd worden.

Waardeberekening

Er is veel onderzoek gedaan naar de waarde van het verbeteren van de Software Development Lifecycle. Deze worden gebruikt om te bepalen hoeveel waarde er is om voor Ace te investeren in deze verbeteringen. De getallen die in dit hoofdstuk behandeld worden, zijn deels gebaseerd op feiten, deels op speculatie en deels op de onderzoeken van Google. Zie de referentie hieronder voor waar de getallen vandaan komen:

[1] Gegevens uit rapport of data van Ace

[2] Google Onderzoek

[3] Speculatie

Return on Investment & Payback Period

De volgende begrippen zijn relevant voor deze berekeningen

- Potential Return = winst na het doorvoeren van geadviseerde verbeteringen
- Investment = kosten voor het doorvoeren van geadviseerde verbeteringen
- Payback Period = aantal dagen die het duurt tot winst begint te ontstaan

De afbeelding hieronder geeft een samenvatting van de Return on Investment en Payback Period. De Potential Return wordt concreet uitgerekend in dit hoofdstuk.

Ace DORA Gap Analyse

De variabelen zijn als volgt ingevuld:

- Hours = [redacted]
- Margin of Uncertainty = [redacted]
- Hourly Fee = [redacted]

Voordat potential return uitgerekend kan worden, moeten eerst de volgende drie variabelen berekend worden:

- Value of rework recovered
- Value lost from new features
- Cost of downtime reduction

Deze variabelen worden in de volgende 3 hoofdstukken berekend.

Cost of Unnecessary Rework Avoided Per Year

Cost of Unnecessary Rework Avoided Per Year kan berekend worden met de volgende formule:

2 Bron: https://services.google.com/fh/files/misc/whitepaper_roi_of_devops_transformation_2020_google_cloud.pdf#page=12

Het doel voor een software team volgens DORA elite performance teams (2024/2023) is te richten op 5% change failure rate. [redacted]

- Technical staff size = [redacted]
- Average Salary = [redacted]
- Benefits multiplier = [redacted]
- [redacted]

[redacted]

Potential Revenue from Reinvestment

Potential Revenue from Reinvestment kan als volgt berkend worden:

3 Bron: https://services.google.com/fh/files/misc/whitepaper_roi_of_devops_transformation_2020_google_cloud.pdf#page=19

[redacted]

Revenue generating features berekenen we als volgt:

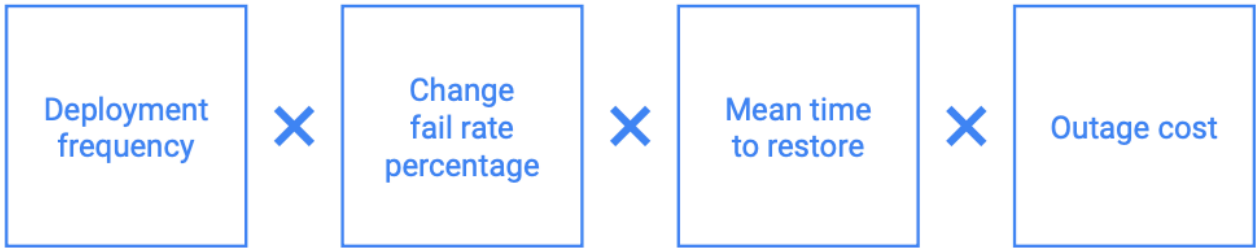
4 Bron: https://services.google.com/fh/files/misc/whitepaper_roi_of_devops_transformation_2020_google_cloud.pdf#page=19

- Frequency of experiments per line of business = [redacted]
- Lines of business in the organization = [redacted]
- Idea success rate = [redacted]
- Idea impact = [redacted]
- Product business size = [redacted]

[redacted]

Cost of Downtime

Cost of Downtime op jaarbasis kunnen we berekenen met de volgende formule:



5 Bron: https://services.google.com/fh/files/misc/whitepaper_roi_of_devops_transformation_2020_google_cloud.pdf#page=25

Gegevens met huidige performance:

- Deployment frequency = [redacted]
- Change failure rate resulting in downtime = [redacted]
- Mean time to restore = [redacted]

- Outage cost = [redacted]

[redacted]

Gegevens met verbeterde performance:

- Deployment frequency = [redacted]
- Change fail rate = [redacted]
- Mean time to restore = [redacted]
- Outage cost = [redacted]

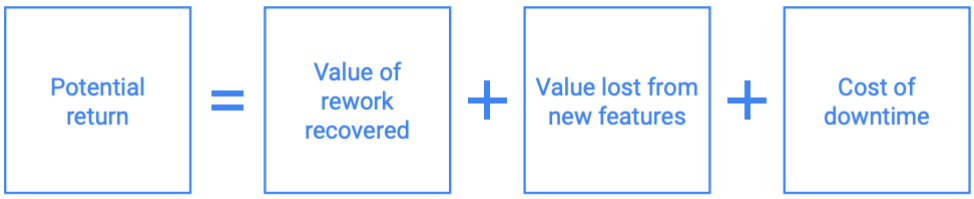
[redacted]

Door de huidige performance en verbeterde performance met elkaar te vergelijken krijgen we het volgende:

[redacted]

Potential Return

Door nu alle returns uit de vorige berekeningen bij elkaar op te tellen krijgen we de potential return.



6 Bron: https://services.google.com/fh/files/misc/whitepaper_roi_of_devops_transformation_2020_google_cloud.pdf#page=29

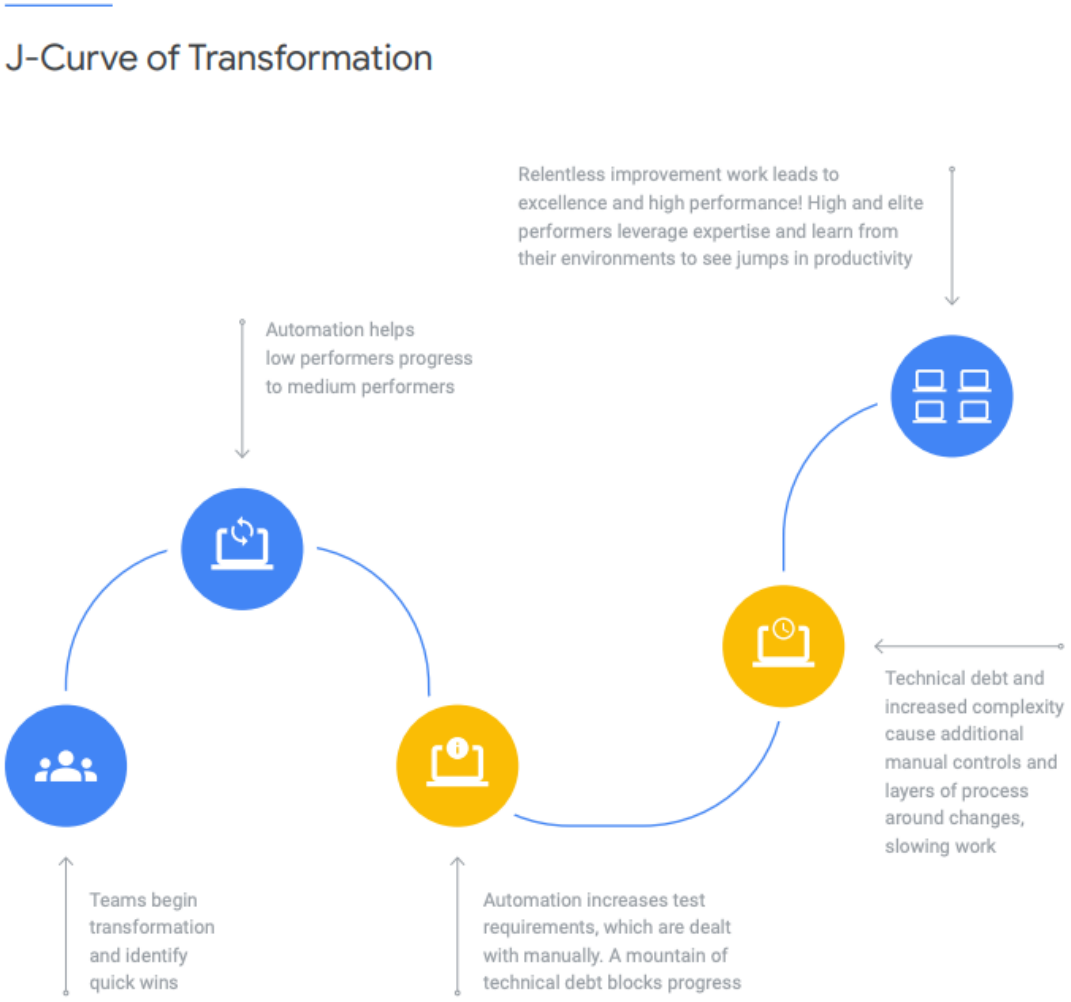
- Value of rework recovered = [redacted]
- Value lost from new features = [redacted]
- Cost of downtime reduction = [redacted]

[redacted]

Wat belangrijk is om hierbij te onthouden, is dat dit deels speculatie is. De cijfers zijn mogelijk niet allemaal representatief voor Ace. Het belangrijkste is echter dat het aantoont dat investeren in verbetering uiteindelijk terug te zien is in financiële winst.

Verwachtingen

Realistische verwachtingen zijn een must bij het uitvoeren van een transformatie gefocust op de SDLC. Dit omdat de praktijk bewijst dat er zonder toewijding ook veel dalen plaatsvinden tijdens het verbeteren van de SDLC-processen. Zie hiervoor de J-Curve of Transformation:



7 Bron: https://services.google.com/fh/files/misc/whitepaper_roi_of_devops_transformation_2020_google_cloud.pdf#page=36