

KU LEUVEN

1 MA INGENIEURSWETENSCHAPPEN:
COMPUTERWETENSCHAPPEN

BEDRIJFSERVARING: COMPUTERWETENSCHAPPEN

**Stageverslag bedrijfservaring
Alcatel-Lucent**

Author:

Helena BREKALO

September 7, 2015

Gegevens student: Helena Brekalo
1e master Computerwetenschappen (Veilige software)

Gegevens bedrijf:

Alcatel-Lucent
Copernicuslaan 50
2018 Antwerpen

Gegevens stagebegeleider:

Bart Hemmeryckx-Deleersnijder
E-mail: bart.hemmeryckx-deleersnijder@alcatel-lucent.be
GSM-nummer: +32 472 95 00 65
Vast nummer binnen het bedrijf: +32 3 240 8525

Stageperiode:

1 juli - 7 augustus
7 september - 11 september

Abstract

Dit verslag beschrijft de opgedane kennis en ervaring tijdens mijn 6 weken- niet mijn?

durende stage bij Alcatel-Lucent in Antwerpen. Het doel tijdens de stage was om een dashboard te ontwerpen en implementeren voor het *continuous integration*-systeem Jenkins en op deze manier “Jenkins-awareness” te creëren.

Het dashboard is succesvol ontwikkeld met behulp van Dashing en toont het overzicht van een reeks zelfgekozenen *builds*. De ontwikkeling is gebeurd in Javascript, Coffeescript, SCSS, Ruby en HTML op Linux. Het dashboard is uiteindelijk in werking genomen.

Het team in Antwerpen zorgde voor enorme betrokkenheid gedurende de stage, door te tonen hoe alle onderdelen van het *agile* proces in zijn werk gaan.

beter maken

Contents

1 Inleiding	3
2 Het bedrijf: Alcatel-Lucent	3
3 Stage	4
3.1 Afdeling: Motive Network Analyzer-Copper	4
3.1.1 Werkmethoden	5
3.2 Stage-opdracht	7
3.2.1 Sprint 1: Onderzoek en literatuurstudie	9
3.2.2 Sprint 2: Ontwikkeling dashboard	9
3.2.3 Sprint 3: Feedback en verbetering	11
3.3 Resultaat	11
3.4 Initiatieven voor stagiairs	13
4 Relatie stage en opleiding	14
5 Kritische reflectie competenties	15
6 Besluit	16
Appendix A Literatuurstudie	17
Appendix B Enquête 1	21
Appendix C Enquête 2	23
Appendix D Bedrijfservaring: competenties	24

1 Inleiding

De probleemstelling, waarvoor tijdens de stage een oplossing ontworpen werd, gaat als volgt. Op de afdeling Motive Network-Analyzer Copper (NA-C) bij Alcatel-Lucent in Antwerpen doet men aan *continuous integration*, waarbij gebruik gemaakt wordt van de *tool* Jenkins. Deze biedt een dashboard om het overzicht van de *build* statussen weer te geven, maar de software-ontwikkelaars hebben niet de neiging deze te bekijken op de televisie die aanwezig is. Het doel van de stage was om “*Jenkins-awareness*” te creëren en een dashboard te ontwikkelen dat een overzicht toont van de *builds* waar aandacht aan besteed moeten worden. Op de afdeling NA-C werkt men volgens het *agile*-principe van *scrum*, waar ook gebruik van gemaakt werd bij het stageproject. Zo werd er een project voor de stage aangemaakt en werden er tweewekelijkse sprints ingepland en evaluaties uitgevoerd.

Om tot het gewenste resultaat te komen is er tijdens de eerste sprint een literatuurstudie uitgevoerd om een idee te vormen van hoe een goed dashboard eruit moet zien. Tijdens de tweede sprint is het dashboard ontwikkeld, met behulp van het *framework* Dashing. Aan het einde van de tweede sprint is deze geëvalueerd door middel van een demonstratie voor de teams in Antwerpen, Chennai en Bangalor. Op basis van de feedback die hierop ontvangen werd via een enquête zijn veranderingen doorgevoerd gedurende de derde sprint. In het tweede deel van de derde sprint is het dashboard in gebruik genomen op de Jenkins-server van NA-C in Antwerpen.

De voornaamste resultaten zijn het dashboard dat succesvol ontwikkeld is, door te werken via de *scrum*-methode. De talen die hierbij gebruikt zijn, zijn Javascript, Coffeescript, (S)CSS, HTML en Ruby. Het dashboard is voorzien van documentatie en er zijn handleidingen geschreven om hergebruik te vergemakkelijken.

Dit verslag is als volgt ingedeeld. In Sectie 2 wordt het bedrijf Alcatel-Lucent voorgesteld. De derde sectie behandelt de stage (Sectie 3), waarin de afdeling waar de stage doorging wordt voorgesteld (Subsectie 3.1), er wordt uitgelegd volgens welke methoden er wordt gewerkt (Subsectie 3.1.1). De stageopdracht wordt uitgelegd in Subsectie 3.2. Het resultaat wordt besproken in Subsectie 3.3. Gedurende de stage waren er ook initiatieven voor stagiairs, wat besproken wordt in Subsectie 3.4.

Sectie 4 legt de link tussen de stage en de opleiding informatica. De kritische reflectie van de competenties die getracht werden verbeterd te worden is te vinden in Sectie 5. Sectie 6 bevat het besluit en de appendices zijn te vinden in Appendix A, B, C en D.

2 Het bedrijf: Alcatel-Lucent

Alcatel-Lucent is een Frans bedrijf met de hoofdzetel in Frankrijk en vestigingen in Amerika, Azië, het Midden-Oosten, Afrika en Europa, waaronder een vestiging in Antwerpen.

De eerste funderingen van Alcatel-Lucent werden gelegd in 1869, met de opstart

ref naar
andere ap-
pendices
toevoegen

van Gray and Barton in Cleveland, Ohio, waarna ze zijn hernoemd naar Western Electric Manufacturing Company. Ongeveer tien jaar later zijn ze overgenomen door American Bell. In 1925 werd Bell Telephone Laboratories gevormd door de samenvoeging van the Western Electric Research Laboratories en een deel van het ingenieursdepartement van AT&T. In deze Bell Labs zijn enkele belangrijke uitvindingen gedaan, zoals die van de transistor. Uit veel van deze uitvindingen heeft Bell Labs nooit een commercieel voordeel gehaald, maar ze waren wel bepalend voor het industriële landschap. In de jaren 1980 werd Alcatel gevormd door de samenvoeging van CIT-Alcatel en Thomson Télécommunications. Aan het begin van de jaren '90 werd Compagnie Générale d'Electricité hernoemd naar Alcatel Alsthom, om aan het einde van de jaren '90 hernoemd te worden naar Alcatel. In 2006 zijn Alcatel en Lucent Technologies gefuseerd, om de huidige naam van Alcatel-Lucent aan te nemen. In 2015 zijn ze ten slotte overgenomen door Nokia, waarbij de overname en hernoeming naar Nokia effectief zal worden in 2016. Door deze overname zal Nokia het tweede grootste bedrijf in draadloze communicatie zijn, op het Zweedse Ericsson na. Op deze manier wordt de concurrentie met het Chinese Huawei vergroot op het gebied van draadloze communicatietechnologie.

Alcatel-Lucent kan meer dan 250 telecomoperatoren wereldwijd tot zijn klanten rekenen, waaronder het Amerikaanse AT&T, Verizon en het Belgische Proximus. Bij de eindgebruiker zijn ze niet echt gekend voor hun producten en diensten, maar binnen de telecomwereld zijn ze een vaste waarde.

Alcatel-Lucent heeft verschillende afdelingen, zijnde het *core networking segment*, wat IP routing, transport en platforms omvat en het *access segment*, wat *wireless* en *fixed access* omvat, alsook *licensing* en *managed services*. In Antwerpen zijn er de afdelingen *Fixed Network*, waarvan het hoofdkwartier in Antwerpen gevestigd is, IP routing R&D, Cloud, Network Functions Virtualization, Software-Defined Networking, R&D en IP platforms R&D.

De stage ging door bij Motive Network Analyzer-Copper (Motive NA-C). Zij staan ervoor in om het gebruik van het kopernetwerk te optimaliseren en fouten op de lijn te detecteren en verhelpen.

Naast deze afdelingen is er ook nog Bell Labs aanwezig in Antwerpen, waar men aan innovatief onderzoek doet samen met universiteiten en andere onderzoeksdepartementen. Ze reiken ook jaarlijks prijzen uit aan theissen van afstuderende studenten, om onderzoek te stimuleren.

slecht verwoord?

3 Stage

3.1 Afdeling: Motive Network Analyzer-Copper

De stage ging door op de afdeling Motive Network Analyzer - Copper (NA-C) bij het team dat onder begeleiding staat van Bart Hemmeryckx-Deleersnijder. Motive NA-C heeft een team in Antwerpen, Chennai en Bangalor, waarbij de ontwikkeling van de software voornamelijk in Antwerpen gebeurt en het testen in Chennai en Bangalor. Er zijn altijd ook enkele testers van Chennai en Bangalor in Antwerpen, om de communicatie tussen de teams te verbeteren en de samenwerking te vergemakkelijken.

Motive NA-C staat in voor het analyseren en onderhouden van de koperlijnen. Hun software zorgt ervoor dat de bandbreedte zo optimaal mogelijk ge-

bruikt wordt en verandert dit gebruik dynamisch naargelang de beschikbare bandbreedte varieert. Indien er fouten optreden op de lijn, kan hun software aanduiden waar op de lijn deze zich waarschijnlijk bevindt en waaraan de fout mogelijks te wijten is. Manieren om tot een oplossing te komen wordt samen met de netwerkanalyse voorzien.

3.1.1 Werkmethoden

Scrum Het team van NA-C werkt volgens de *scrum*-methode, wat een voorbeeld is van een *agile* werkmethode, waarbij men ervan uitgaat dat de vereisten voor het project kunnen en zullen veranderen gedurende het project. Deze werkmethode staat tegenover de “klassieke” watervalmethode, waarbij eerst de vereisten worden opgesteld om die vervolgens te implementeren en evalueren. Hierna komt men in de onderhoudsfase van het systeem. Het probleem met deze manier van werken is dat de communicatie tussen de klant en het bedrijf niet optimaal verloopt, gezien de klant vaak zijn mening over de vereisten van het project nog herziet nadat de eerste versie van de software is uitgebracht. De veranderingen kunnen dus vaak pas tijdens de onderhoudsfase worden doorgevoerd. Als de aangepaste vereisten pas in de onderhoudsfase ontdekt en aangekaart worden, is het heel kostelijk om (grote) veranderingen door te voeren in het reeds ontwikkelde systeem.

Bij een *agile* werkmethode, waar *scrum* een onderdeel van is, gaat men er van uit dat de klant op voorhand niet perfect kan weten wat hij wil en dat hij dus de vereisten zal aanpassen. Hierop wordt dan ingespeeld tijdens het ontwikkelingsproces, door dit niet zoals het watervalmodel te zien, maar te werken in ontwikkelingsfasen van korte perioden, die gewoonlijk één tot vier weken duren. Op deze manier kunnen veranderingen in de vereisten snel opgepikt en doorgevoerd worden, wat het ontwikkelingsproces minder kostelijk maakt.

De term “*scrum*” is afkomstig vanuit rugby, waarbij de twee teams voorovergebogen tegen elkaar leunen, met de hoofden bij elkaar. De bal wordt dan in deze groep gegooid, waarop ze proberen om de bal als eerste in hun bezit te krijgen. De gelijkenis met *scrum* bij software ontwikkeling is dat het team heel nauw samenwerkt, inspeelt op veranderende omstandigheden en zo samen tot een doel komt. De samenwerking uit zich onder andere in dagelijkse *stand-up meetings*, die telkens rond hetzelfde uur plaatsvinden, ongeacht of iedereen aanwezig is of niet. Tijdens deze *meetings* vertelt iedereen wat hij/zij de dag ervoor heeft gedaan, welke problemen er zijn opgetreden en of er daar hulp bij nodig is en wat de planning voor de komende dag is. Een foto van een van de stand-up meetings van het NA-C team is te zien in Figuur 1. Dankzij de dagelijkse *stand-up meetings* is iedereen op de hoogte van wie waarmee bezig is en kunnen problemen snel aangekaart en opgelost worden. Bij het NA-C team in Antwerpen maakt men hierbij ook gebruik van het *agile* dashboard van JIRA¹, waarbij gevisualiseerd wordt wie waarmee bezig is (zie Figuur 2). Indien iemand ergens problemen mee heeft, doet men aan *pair-programming*, waarbij men bij elkaar gaat zitten en samen nadenkt over hoe het desbetreffende probleem opgelost kan worden en dat samen uit te werken.

In het NA-C team zijn er sprints van twee weken, wat wil zeggen dat er

¹<https://www.atlassian.com/software/jira>



Figure 1: Een stand-up meeting van het NA-C team in Antwerpen.

Figure 2: Het JIRA dashboard van het NA-C team in Antwerpen.

elke twee weken een evaluatie plaatsvindt van het werk dat in de afgelopen twee weken verzet is en waar besproken wordt welke veranderingen er naar de volgende sprint toe moeten doorgevoerd worden. Aan het begin van een sprint is er de sprint planning, waarin er bekijken wordt wat er moet gebeuren met welke prioriteit en wie wat gaat doen, op een heel algemeen niveau. Hierbij wordt gebruik gemaakt van *user stories*, die een bepaalde taak omschrijven en de subtaken ervan. Het beschrijft wat de gebruiker nodig heeft bij het uitvoeren van zijn taak en bepaalt op deze manier wat er moet ontwikkeld worden om de gebruiker hierbij te helpen. Het omschrijft het “wie”, “wat” en “waarom” van deze vereisten op hoog niveau. Een voorbeeld van een *user story* is te zien in Figuur 3. Aan het einde van de sprint is er de sprint *review*, waarbij besproken wordt wat er bereikt is en waarbij eventueel demo’s gegeven worden van wat er gerealiseerd is. Ook is er de sprint *retrospective*, waarbij opgeliijst wordt wat er goed en minder goed is gegaan, welke problemen er (binnen het team) waren en hoe die opgelost kunnen worden.

Figure 3: Een van de *user stories* van het team van NA-C in Antwerpen.

Continuous integration Aangezien er regelmatig *releases* van software zijn, wordt er gebruik gemaakt van *continuous integration*, wat inhoudt dat wanneer iemand klaar is met het schrijven van een stuk code en deze *commit*, alle *builds* en testen hiermee automatisch zullen *runnen* zodat fouten in een vroeg stadium

kunnen gevonden en opgelost worden.

In Antwerpen wordt gebruik gemaakt van Jenkins², een *open-source continuous integration* systeem. Jenkins heeft een dashboard dat een overzicht toont van alle *builds*, maar er kan ook gefilterd worden op zelfgemaakte criteria, zodat enkel de statussen van bepaalde *builds* zichtbaar zijn. Een screenshot van dit dashboard is te zien in Figuur 4. Het overzicht van elke *build* toont een gekleurde bol om aan te geven of de laatste *build* geslaagd (groen of blauw) of gefaald (rood) is. Indien er testen gefaald zijn, dan zal de bol geel kleuren. Er is ook een weerbericht dat toont hoe stabiel de *build* is, waarbij slecht weer duidt op een instabiele *build* en goed weer op een stabiele *build*.

moet dit cursief?

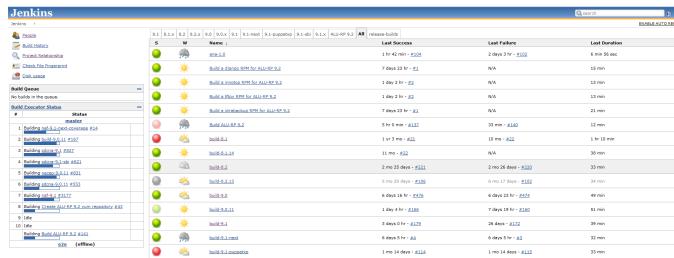


Figure 4: Het Jenkins dashboard, zoals oorspronkelijk gebruikt op de afdeling NA-C in Antwerpen.

Het probleem met dit dashboard is dat het niet overzichtelijk is. De kleur van de bollen trekt wel de aandacht, maar het is vanop een afstand niet duidelijk welke *builds* getoond worden. Het Jenkins dashboard wordt op de werkvloer weergegeven op een grote televisie, maar doordat het zo onoverzichtelijk is, wordt er niet naar gekeken. Zoals op Figuur 5 te zien is, is het zelfs op een groot scherm niet duidelijk welke *builds* er getoond worden.

3.2 Stage-opdracht

De stage-opdracht bestond erin om “Jenkins awareness” te creëren aan de hand van een dashboard dat een beter overzicht toont. Wanneer gefaalde *builds* meer in het oog springen, kan de code sneller herbekeken worden en is er echt sprake van *continuous integration*. Gedurende heel de stage werd betrokkenheid gevraagd bij alles wat het team deed, zoals meedoen met de *stand-up meetings* en het bijwonen van de sprint planning, de sprint *review* en de sprint *retrospective*, om zo een idee te vormen van hoe de *agile* manier van werken toegepast wordt in een bedrijfsomgeving. Dat maakte duidelijk hoe deze manier van werken een team sterker kan maken, gezien iedereen weet wie waarmee bezig is en hoe de korte evaluatieperiodes ervoor zorgen dat er kort op de bal gespeeld kan worden. Voor de stage-opdracht werd een apart project gemaakt, waarbij er ook gewerkt werd volgens de *agile*-methode met tweewekelijks sprints. Er werd voor het project een JIRA-pagina aangemaakt, waarop de vooruitgang werd bijhouden door taken aan te maken die moesten uitgevoerd worden en waar problemen gerapporteerd werden. Een screenshot van dit JIRA-dashboard is te zien in Figuur 6. De eerste sprint bestond uit het maken van een literatuurstudie, te vinden in Appendix A. Hierin wordt onderzocht waarop moet worden bij

goed gezegd?

²<https://jenkins-ci.org/>

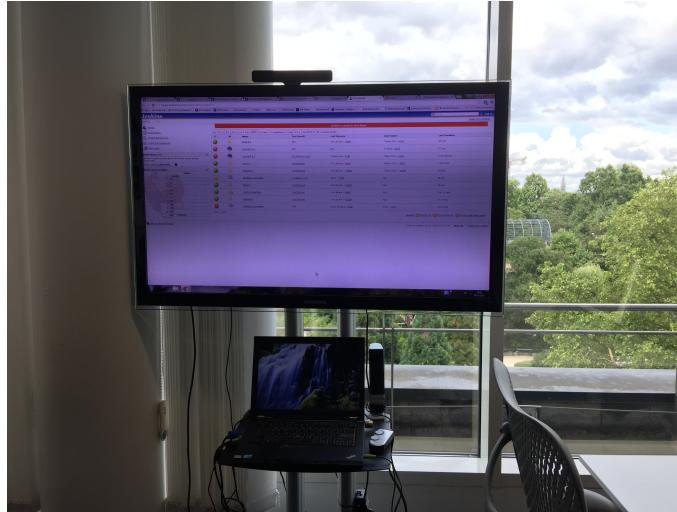


Figure 5: Het originele Jenkins dashboard weergegeven op de televisie op de afdeling van NA-C in Antwerpen.

het maken van een (goed) dashboard en wat er verbeterd kan worden aan het originele Jenkins dashboard. Vervolgens zijn enkele voorstellen voor dashboard-layouts opgemaakt en is er een kleine enquête uitgevoerd binnen het team, om te kijken welke voorkeuren zij hadden. Hierop wordt dieper ingegaan in sectie 3.2.1.

Tijdens de tweede sprint is het dashboard ontwikkeld en voorgesteld aan de NA-C teams in Antwerpen, Chennai en Bangalor. Dit proces is te vinden in sectie 3.2.2.

De derde en laatste sprint bestond uit het voeren van een enquête over de huidige versie van het dashboard, om het vervolgens te verbeteren. In de tweede helft van deze sprint is het dashboard in gebruik genomen op de Jenkins *server* van Alcatel-Lucent. Dit wordt besproken in sectie 3.2.3.

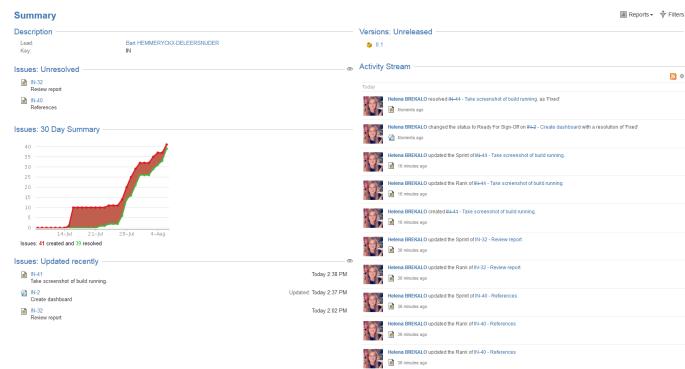


Figure 6: Het JIRA-dashboard voor mijn stage.

3.2.1 Sprint 1: Onderzoek en literatuurstudie

Literatuurstudie Tijdens de eerste sprint is er onderzoek gevoerd naar hoe goede dashboards eruit zien en hoe ze overzichtelijk gehouden kunnen worden. Hierond is een literatuurstudie gemaakt, die gevonden kan worden in Appendix A. Uit deze literatuurstudie bleek dat het voorziene Jenkins dashboard niet voldeed aan veel van de vereisten, aangezien de belangrijkste informatie niet één oogopslag zichtbaar is.

Naast het maken van de literatuurstudie is er ook bekeken welke *tools* er zijn om dashboards te maken, waarbij Dashing³ aangeraden werd door een collega. Na het bekijken van enkele andere *tools* om dashboards te ontwikkelen, is er uiteindelijk voor Dashing gekozen, aangezien het enorm veel flexibiliteit biedt.

Enquête Na het onderzoek en de literatuurstudie is er een voorstel voor dashboardontwerpen opgemaakt die werd voorgelegd aan het team bij NA-C in Antwerpen. Hierbij hoorde een enquête waarbij gekozen kon worden voor verschillende ontwerpen en waarbij opmerkingen gegeven konden worden. Uit die opmerkingen zijn nog nuttige commentaren gekomen, waarbij enkele van die ideeën ook in het dashboard verwerkt zijn. De enquête en de resultaten zijn te vinden in Appendix B.

3.2.2 Sprint 2: Ontwikkeling dashboard

Op het einde van de eerste sprint vond een *sprint review* en *sprint retrospective* plaats, om de gemaakte vooruitgang te evalueren. Deze meeting werd gecombineerd met de sprint planning, waar een eerste versie van het ontwerp en de *tools* voor het dashboard werden voorgesteld.

Gebruikte tools Dashing is een dashboard *framework* dat geschreven is in Sinatra, een *open source* software web applicatie geschreven in Ruby. De ontwikkeling gebeurde in Linux, waarbij gebruik gemaakt werd van Ubuntu 14.04. Op deze PC is een lokale Jenkins-omgeving opgezet, zodat het dashboard tijdens de ontwikkeling hiermee getest kon worden. Het schrijven van de code gebeurde in gedit en debuggen gebeurde via de *terminal*.

De “tegels” op het dashboard worden omschreven als *widgets* waarbij je verschillende soorten *widgets* kan hebben, met elk een bepaalde functionaliteit. Tijdens deze fase van het project werd er gefocust op een *widget* om informatie over Jenkins te tonen. Deze *widget* kan gebruikt worden voor verschillende *builds*, het is niet de bedoeling om voor elke *build* een nieuw *widget* te ontwikkelen, tenzij je andere informatie wilt tonen op de verschillende Jenkins-*widgets*. Tijdens de derde sprint is hier nog een *widget* voor *RSS-feeds* aan toegevoegd, zodat ook het meest recente nieuws gevolgd kan worden.

Gebruikte talen Het schrijven van het dashboard is gebeurd in verschillende talen, hieronder een opsomming welke taal gebruikt werden.

HTML HTML wordt gebruikt om te bepalen hoe de inhoud getoond wordt op de *widget*, dus bijvoorbeeld welk deel als titel zal getoond worden en welke onderdelen als paragrafen weergegeven worden. HTML bepaalt ook

³<http://dashing.io/>

de locatie van de elementen op de *widget*. De informatie die getoond wordt, wordt via Batman Bindings⁴ uit de JSON data van de Jenkins server *geparsed*.

goed woord?

(S)CSS CSS wordt gebruikt om de tekst uit het HTML-bestand op te maken (grootte van de tekst, stijl, kleur,...), waarbij SCSS (Sassy CSS) ervoor zorgt dat je ook variabelen kan gebruiken, wat niet mogelijk is in standaard CSS. Op deze manier kan code van een bepaalde opmaak hergebruikt worden via variabelen.

Javascript Javascript werd gebruikt in de Batman Bindings, die ervoor zorgen dat de data die van de website *geparsed* wordt, dynamisch gebruikt kan worden in de HTML-bestanden. Op zich is de Batman Bindings-*library* klaar om te gebruiken, maar om een tijdstip te *parsen* (dat werd weergegeven als Unix *timestamp*), is gebruik gemaakt van de moment.js-*library*.

Coffeescript Coffeescript is een taal die transcompileert naar Javascript. Het .coffee-bestand voor een *widget* specificeert hoe de data, die via het HTML-bestand opgevraagd wordt, weergegeven wordt. Dit bestand was van belang om onder andere het juiste weerbericht te tonen. Het coffeescript parset de data die via de Batman Bindings werd opgehaald en geeft aan hoe die doorgegeven wordt aan het HTML-bestand. Zonder het .coffee-bestand zou er enkel *plain text* (mits wat opmaak door CSS) te zien zijn op de *widget*.

Ruby Dashing maakt gebruik van Embedded Ruby om te bepalen welke tegels getoond worden op het dashboard en van *plain Ruby* om de data op te halen en te *parsen* vanuit de juiste plaats (nl. de Jenkins server) via de JSON-data die werd voortgebracht hierdoor. Ook wordt hier de link gelegd tussen de *build jobs* die getoond kunnen (niet per sé zullen⁵) worden op het dashboard.

JQuery JQuery wordt gebruikt in het coffeescript bestand om CSS-gewijs klassen of ID's te gebruiken van elementen waarop een bepaalde actie moet uitgevoerd worden. Er is van JQuery gebruik gemaakt om de *build* meter te doen verschijnen en verdwijnen bij het starten of aflopen van een *build* en om het weerbericht en de status van de *build* te tonen.

JSON JSON wordt gebruikt om de data uit van de Jenkins server te *parsen*, zodat die opgevraagd kan worden in het Ruby bestand.

Er is gebruik gemaakt van Codecademy⁶ en online *tutorials* om basiskennis te verwerven van de gebruikte talen.

De verschillende bestanden worden als volgt gebruikt:

- Een *widget* wordt opgesteld door middel van een .HTML-bestand, een .scss-bestand en een .coffee-bestand.

⁴<http://batmanjs.org/>

⁵Welke build jobs getoond worden, wordt bepaald door de Embedded Ruby file.

⁶<https://www.codecademy.com/>

- Een Embedded Ruby bestand geeft via HTML-*blocks* aan welke *build* jobs getoond worden op het dashboard.
- Een *plain* Ruby bestand doet de *mapping* van de JSON-data op de variabelen in het dashboard en definiëert de namen van de *build* jobs die getoond kunnen worden.

Sprint Review Op het einde van de tweede sprint is de eerste versie van het dashboard gedemonstreerd voor het team in Antwerpen, Bangalore en Chennai. Er kon een werkend dashboard getoond worden dat informatie weergeeft over de Jenkins *build* jobs die op een lokale *server* draaiden. De feedback hierop was positief en enkele vragen van mensen in de verschillende teams konden beantwoord worden.

goed
gezegd?

3.2.3 Sprint 3: Feedback en verbetering

Na de sprint review is de Alcatel-Lucent Jenkins-server geïntegreerd in het dashboard, wat nog anderhalve dag van *debugging* vroeg, gezien de Jenkins-server nu niet lokaal draaide, maar op een andere server (nl. de Jenkins-server van het NA-C team in Antwerpen). Ook gebruikt men bij Alcatel-Lucent een andere versie van Jenkins, waardoor bepaalde sleutelwoorden waarop *geparsed* wordt anders heten.

Na de sprint *retrospective* van sprint 2 en de sprint planning van sprint 3 is er nog een enquête uitgevoerd om positieve en negatieve commentaren te verzamelen. De resultaten van deze enquête zijn te vinden in Appendix C. Een van de uitbreidingen die zijn gemaakt is het tonen van een *RSS-feed* dat nieuws weergeeft. Ook is het dashboard aangepast zodat de meter van de *build* status verdwijnt wanneer de *build* compleet is. Enkele screenshots van het dashboard uit de derde sprint zijn te zien in Sectie 3.3.

Ingebruikname

over
vertellen

3.3 Resultaat

Resultaat na sprint 2 De eerste versie van het dashboard is te zien in Figuur 7. Het dashboard in zijn huidige vorm is gemaakt om getoond te worden op een HD-scherm met 16:9-ratio. Het is de bedoeling dat er zes tegels naast elkaar kunnen staan en dat er drie rijen zijn, wat maakt dat de status van 18 *build* jobs gelijktijdig getoond kunnen worden. Dit bleek ruim voldoende voor het NA-C team, aangezien er altijd maar op enkele *builds* tegelijkertijd gefocust wordt.

Figuur 8 toont de tegels in meer detail. De titel is de naam van de *build* waarover de tegel informatie verschafft. Het weerbericht toont de stabiliteit van de *build*, of, indien er testen gefaald zijn, de stabiliteit van de testen. De tekst eronder geeft aan hoe stabiel de *builds* of de testen zijn, de meter geeft aan hoe ver de *build* gevorderd is het tijdstip daaronder toont wanneer de laatste *build* uitgevoerd is.

Er zijn verschillende achtergrondkleuren gebruikt die de status van de laatste *build* weergeven:

- Grijstint: de *build* is nog bezig, het weerbericht toont de stabiliteit van de *builds* van voor degene die nu bezig is.

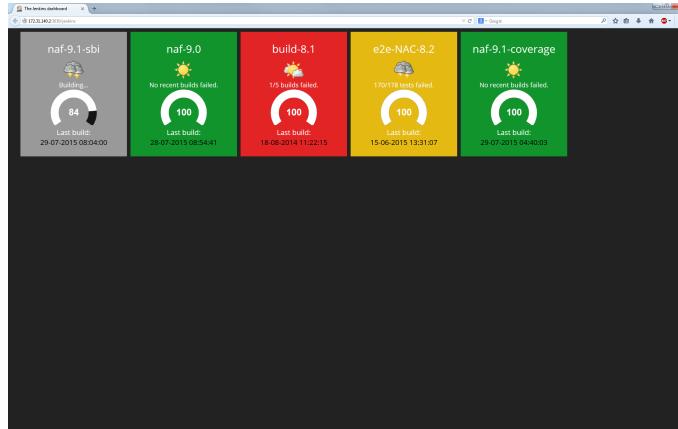


Figure 7: Eerste versie van het dashboard.

- Rood: de laatste *build* is gefaald.
- Geel: één of meerdere testen zijn gefaald.
- Groen: de laatste *build* is geslaagd en alle testen zijn geslaagd.

Als er gekeken wordt naar het voorstel opgemaakt in de literatuurstudie is de achtergrondkleur daar niet aanwezig (de *build status* wordt er weergegeven met een gekleurde bol), maar de enquête bij NA-C in Antwerpen toonde dat iedereen te vinden was voor de gekleurde achtergrond.

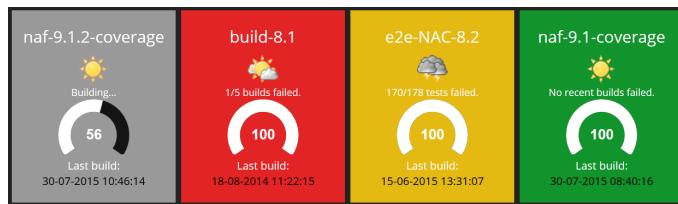


Figure 8: Gedetailleerd overzicht dashboard.

Resultaat na sprint 3 Na de tweede enquête (te vinden in Appendix C) zijn nog enkele veranderingen doorgevoerd. De belangrijkste zijn:

- De meter verdwijnt wanneer de *build* niet bezig is, zo leidt deze de aandacht niet onnodig af.
- Er is een *RSS-feed* toegevoegd voor nieuwsitems.
- Er kunnen zones van *widgets* gecreëerd worden, om te tonen welke samen horen. In Figuur 9 is dit gedaan met de nieuwsitems, maar het kan dus ook perfect met de *build widgets*.

Er zijn ook twee nieuwe kleuren van *widgets* toegevoegd:



Figure 9: Finale versie dashboard.



Figure 10: Finale dashboard op televisie.

- Zwart: de desbetreffende *build* is uitgeschakeld (*disabled*). Er wordt op deze *widget* geen weerbericht getoond, wel het tijdstip waarop de laatste *build* is uitgevoerd.
- Lichtblauw: deze kleur is bedoeld voor de nieuws*widget*, om deze duidelijk te kunnen onderscheiden van de Jenkins *widgets*.

Het finale resultaat is te zien in Figuur 9. Op Figuur 10 is het dashboard te zien op de televisie van de afdeling NA-C. Figuur 11 toont de finale versie van het dashboard in meer detail.



Figure 11: Gedetailleerd overzicht van de finale versie van het dashboard.

3.4 Initiatieven voor stagiairs

Alcatel-Lucent organiseerde in de zomer van 2015 een initiatief om alle stagiairs met elkaar in contact te brengen tijdens speciale info-events en *meetings*. Op deze manier konden stagiairs van verschillende afdelingen met elkaar bespreken hoe de stage op hun afdeling verliep en zo ervaringen met elkaar uitwisselen. Er zijn verschillende contactmomenten georganiseerd, meestal tijdens de middagpauze. Er waren verschillende bijeenkomsten, waaronder samen lunchen om kennis te maken en een gesprek met Joeri Veldeman, begonnen als doctor in de *electrical engineering* bij Alcatel-Lucent en nu hoofd van de afdeling van *human resources*. Zijn ongewone carrièrepad werd besproken en hij er werd informeel gesproken over de verdere studietrajecten van de stagiairs. Verder is er een infosessie geweest over ALU Labs, waarin enkele van de zaken aan bod kwamen waarin Alcatel-Lucent investeert en onderzoek bij uitvoert. Er was ook een infosessie over *fixed networks*, waarbij uitleg gegeven werd over wat het team in Antwerpen allemaal verwezenlijkt en waarbij een bezoek gebracht werd aan een

weglaten?

van de labo's.

Dit alles maakte dat de stage niet alleen een verrijking van kennis binnen de afdeling werd, maar ook daarbuiten, waarbij contact met andere studenten gelegd werd.

4 Relatie stage en opleiding

Het doel van de stage was om een dashboard te maken zodat de *build* status van de verschillende onderdelen van projecten overzichtelijk getoond kunnen worden. Binnen deze stage werd ik bovendien ook betrokken bij alle bijeenkomsten met het team, om zo te ervaren hoe het er in een werkomgeving aan toe gaat. Op deze manier heb zijn extra facetten aan bod gekomen die tijdens de opleiding aan bod zijn gekomen. Zo werkt men op de afdeling NA-C volgens de *agile*-methode, waar scrum een onderdeel van is, met elke dag *standup meetings* en tweewekelijkse sprints. Dit is iets dat in de lessen van Software-Ontwerp aan bod is gekomen, waarvan tijdens de stage bleek dat het echt zijn nut heeft. De korte sprints en regelmatige *reviews* zorgen ervoor dat problemen niet te lang kunnen blijven bestaan en omdat er bijna dagelijks gesproat wordt over wie waarmee bezig is, kunnen mensen elkaar vlot helpen en weet iedereen wie waarmee bezig is. Doordat er gebruik gemaakt wordt van *continuous integration* zijn (*merge*)fouten snel opgespoord en opgelost. Voor de stageopdracht werd een eigen scrumproject voorzien, waarbij ook over sprints geïtereerd werd. Die manier van werken is vergelijkbaar met deze tijdens de P&O-sessies van de bachelorproef, waarbij we om de paar weken een bepaalde functionaliteit moesten kunnen demonstreren. Op de afdeling NA-C wordt er regelmatig aan pair-programming gedaan, iets wat ook aangeraden werd tijdens Software-Ontwerp. De stage-opdracht was onafhankelijk van de projecten bij NA-C, maar bij vragen bij het project is er wel aan *pair-programming* gedaan. Het dashboard dat ontwikkeld moest worden moest niet een kant-en-klaar dashboard zijn waar zelf info aan toegevoegd werd, maar één waarbij moest nagedacht worden over het design en de implementatie van de code om het dashboard te realiseren. Dit werd behandeld in de lessen van Software-Ontwerp.

Er werd gebruik gemaakt van een bestaand *framework*, maar er is nog een groot deel code bijgeschreven. Omdat er gebruik gemaakt werd van bestaande code, werd het duidelijk hoe belangrijk goede documentatie is, gezien de bestaande code op zich nauwelijks/niet gedocumenteerd was, wat het begrijpen van de code moeilijker maakte. Het schrijven van (goede) documentatie is iets waar tijdens de lessen van Objectgericht Programmeren de nadruk op lag. De documentatie op de websites van het *framework* en de *plugins* was vaak onvolledig, wat het opzoeken van functionaliteiten moeilijker maakte. Om die redenen is er tijdens het ontwikkelen van het dashboard eigen documentatie geschreven. Om hergebruik en aanpassing te vergemakkelijken, zijn er handleidingen geschreven op de wiki van Alcatel-Lucent.

Binnen de opleiding wordt er altijd gewerkt in kleine groepjes, meestal van twee mensen, maximaal in groepen van zes mensen. Op de afdeling NA-C werkt men in een team van ongeveer 10 mensen, wat het managen van het team nog moeilijker maakt dan bij een groep van slechts enkele mensen. Toch is er getoond dat dit perfect mogelijk is indien je de juiste *tools* gebruikt (bij NA-C maakt men gebruik van Confluence en JIRA) en indien er goed gecommuniceerd wordt.

puntjes maken

Tijdens de lessen van Informatica werktuigen zijn Ruby, Linux en LaTeX aan bod gekomen. Ruby bleek nuttig om de (Embedded) Ruby voor het dashboard te schrijven, waar de kennis van Linux ervoor zorgde dat werken met Linux in het algemeen vlot ging, gezien het dashboard in Linux is ontwikkeld. LaTeX is gebruikt om de literatuurstudie te maken en het verslag op te stellen.

Doordat er in de bachelor informatica al een heel aantal programmeertalen gezien werden, was het aanleren van andere (opmaak)talen veel makkelijker. Het schrijven van (wetenschappelijke) verslagen was een groot onderdeel tijdens Wetenschapscommunicatie en tijdens P&O, wat hielp bij het schrijven van de literatuurstudie en het stageverslag.

Waar versiecontrole tijdens het project van Software-Ontwerp werd aangeraden, maar niet verplicht was, werd het bij P&O wel verplicht. Hiervan is gebruik gemaakt bij het ontwikkelen van het dashboard (zij het aan de late kant, zie de kritische reflectie van de competenties in Sectie 5) tijdens een groot deel van de stage.

evt. tekst
bijvoegen

5 Kritische reflectie competenties

Het originele document met competenties die ik wenste te verbeteren is te vinden in Appendix D.

- Technische competenties die ik hoopte te verbeteren:

Bijleren over *version control* Er wordt op de afdeling NA-C gewerkt met Git, waar ik ook tijdens mijn opleiding al mee in aanraking ben gekomen. Tijdens de eerste fase van de ontwikkeling van het dashboard heb ik geen gebruik gemaakt van Git, hoewel dit zeker nuttig geweest zou zijn. Eens de eerste versie van het dashboard klaar was, is er gebruik gemaakt van Git⁷, maar dit had dus veel eerder moeten gebeuren. Er is gebruik van gemaakt bij het toevoegen van documentatie en bij het maken van veranderingen tijdens de derde sprint.

Ondanks het te laat beginnen gebruiken van versiecontrole, zijn mijn competenties hiermee wel verbeterd, gezien ik Git nu ook via de Linux *command line* heb leren gebruiken en niet alleen in de grafische *user interface* op Windows.

dit beter
vertellen?

Leren werken met Jenkins Op de afdeling NA-C wordt gewerkt met het *continuous integration* systeem Jenkins, dat *builds* automatisch *runt* nadat er een nieuwe versie van de software *gocommit* is. Tijdens de stage was het duidelijk hoe nuttig dit is: er werd op een gegeven moment een fout gemaakt, die ook nog eens onvoldoende getest bleek, maar men heeft die heel snel ontdekt en binnen de 2 dagen kunnen oplossen. Moest er gewacht zijn met *mergen* tot vlak voor de *release*, was men waarschijnlijk veel meer problemen tegengekomen. *Continuous integration* is vooral handig in teams met meerdere mensen, het was dus van pas gekomen tijdens P&O of Software-Ontwerp.

dat laatste
weglaten?

Mij inwerken in een reeds lopend project Ik moest niet meewerken aan de code van het NA-C team zelf, maar heb mij wel moeten in-

⁷De code is te vinden op <https://github.com/HelenaCat/jenkins-dashboard>

werken in de code van het dashboard. Zoals hierboven vermeld, was dit niet altijd makkelijk, maar het heeft er wel voor gezorgd dat ik enorm veel heb bijgeleerd, net omdat ik heel veel zelf moest uitzoeken. Eens de eerste versie van het dashboard klaar was, kende ik de code enorm goed, dus nieuwe veranderingen konden zeer snel geïmplementeerd worden.

- Persoonlijke competenties die ik hoopte te verbeteren:

Leren werken in grote teams In het NA-C team in Antwerpen werken er een tiental mensen, in Bangalore en Chennai zijn er teams van gelijkaardige grootte. Doordat er dagelijks een *standup meeting* was met de mensen van het team in Antwerpen, was iedereen goed op de hoogte van wie wat deed en doordat men met sprints van twee weken werkte, kwamen problemen snel boven. Dit is iets dat ik zeker meeneem naar projecten in de toekomst toe, ik heb ervaren hoe al deze zaken kunnen helpen in het beter doen verlopen van een project.

Plannen op lange termijn Aan het begin van de stage werd me meteen gezegd dat ik redelijk vrij gelaten ging worden in mijn planning, maar er werd wel gevraagd om er een op te stellen, zodat ik ook in sprints kon werken. Ik heb zelf voorstellen mogen doen en aan mijn eigen tempo (maar toch binnen de termijnen van de sprints) kunnen werken. Als er grote dingen te doen stonden, plande ik daar altijd genoeg tijd voor in, zodat ik uiteindelijk “te vroeg” klaar was en aan verbeteringen kon werken. Dit is een van de competenties die ik zowel in het tweede semester van mijn laatste bachelorjaar heb aangescherpt, als nu op de stage, dankzij de goede begeleiding van Bart.

6 Besluit

De stage ging voor 6 weken door bij Alcatel-Lucent in Antwerpen. De opdracht was het ontwerpen van een dashboard om de *build* status van het *continuous integration* systeem Jenkins te tonen. Het ontwikkelen van het dashboard is gebeurd gebruik makende van de *agile* werkmethode, waarbij in sprints van twee weken gewerkt werd.

Het dashboard is succesvol gemaakt binnen de opgelegde tijd met behulp van Dashing, draaiende op een Linux server.

Tijdens de stage is gebruik gemaakt van onder andere Javascript, JQuery, HTML en (S)CSS. Mijn kennis van Ruby en Linux kwam hierbij ook van pas. Onder de goede begeleiding van Bart en dankzij de hulp van de mensen bij NA-C in Antwerpen heb ik grotendeels zelfstandig mijn doelen en het project uitgewerkt en geleerd hoe werken in teams in een bedrijf verloopt.

Appendix A Literatuurstudie

General A dashboard is a tool to quickly show information at a glance. Its purpose is to have the “reader” gather information without having to look things up; everything should be clear by having a quick look at the dashboard. At Alcatel-Lucent, they make use of the Jenkins dashboard, but it doesn’t comply with the given description: in order to see what builds failed/succeeded, you need to find the build you’re looking for and then check its status. It is clear from the -lack of- its use that this way of displaying the information does not suffice, since the TV dashboard is not/barely used but every employer looks up the dashboard on their own PC every once in a while.

The purpose of this literature study is to create Jenkins awareness, so the developers can take a look at the TV dashboard a few times a day and have them see how the project is proceeding.

One of the most important things is that the information displayed on the dashboard can be easily interpreted. This means that you don’t want a cluttered dashboard with too much information so it becomes unclear. Stephen Few has written an excellent article on the topic, with the main conclusions being:

- don’t show too much information,
- keep the information simple,
- colors should be used sparingly or they will mean nothing at all,
- visuals should be useful; not overwhelming or distracting and
- the information represented should be accurate, so you should make sure the information is displayed correctly and with enough context.

This last remark is also made by Matthew Skelton⁸, who shows that if you only show the failed builds, people will start to ignore the ‘alarm signals’ the dashboard displays and perform less well compared to when a context is shown, also showing the successful builds.

Few’s theory is backed up by the people at Geckoboard, who have made a blog consisting of 6 parts, referencing to Few, but also adding some rules of thumb⁹. Every article listed in the appendix refers to Few’s article and some make additional remarks. One of them is that it’s very important to keep your audience in mind, the board of directors will want to see different metrics represented in the dashboard than, say, the people of human resources. In the case of software developers, they’ll want to see metrics about the build status of the different jobs. Related to this is that not every dashboard will need the same layout and look.

In the case of the Jenkins dashboard, charts won’t tell you much, since it doesn’t really tell anything about the status of the last build. Below, the example of the Jenkins dashboard is further explained and a proposal is made that conforms to the guidelines of a good dashboard.

⁸<http://blog.matthewskelton.net/2013/03/11/what-makes-an-effective-build-and-deployment-radiator-screen/>

⁹<https://www.geckoboard.com/blog/building-great-dashboards-6-golden-rules-to-successful-dashboard-design/>

Improving the Jenkins dashboard The Jenkins dashboard, as seen in Figure 12, displays a list of the builds of a project, together with a colored dot and a “weather report”. The dot displays information about the last build, the color indicating whether it failed or succeeded and a flashing dot meaning the build is still in progress. The color of the dot at that time then refers to the last finished build. The weather report tells you something about the stability of the builds. If the weather looks good, then the build is stable. The worse the weather gets, the less stable the build is. This way, you get an overall view about the stability of the different build jobs.



Figure 12: The Jenkins dashboard.

These two things give you a fairly good overview, but the downside of the dashboard is that you have to be right in front of the screen to see what job the colored dot and the weather report are referring to. The details of the build job are in plain text, so if you want to project it on a TV screen, people will have a hard time knowing what builds are doing well and which ones aren’t. There are plug-ins available that show a dashboard, but they none of them comply with the most important rules described above: they’re either too crowded or use too much color, distracting the viewer from what’s really important.

The proposed images, as seen in Figure 13 and 14 comply with the design problems to be avoided as proposed by Stephen Few:

- Too much complexity
- Too many alert conditions
- Alerts that cannot be differentiated
- Overwhelming visuals
- Distracting visuals
- Inappropriate visual salience
- Mismatch between information and its visual representation
- Indirect expression of measures
- Not enough context

The images show the build stability together with the success or failure of the last build and show the name of the build clearly. It’s also possible to add a timestamp of when the last success and failure were and the last duration at

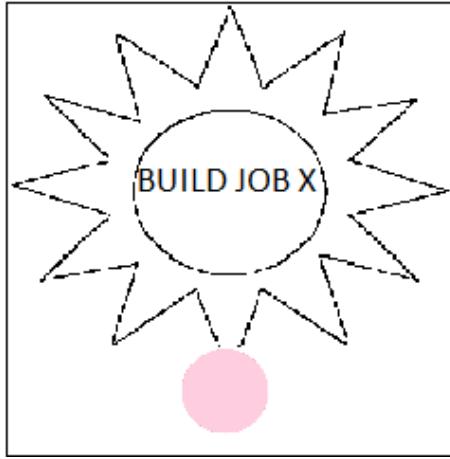


Figure 13: Proposal 1.

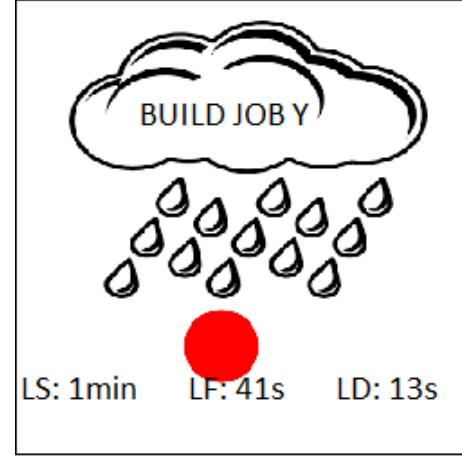


Figure 14: Proposal 2.

the bottom of the tiles, as shown in Figure 14. If you have color blind people in your team (as Few pointed out might happen), you could replace the green dot by a very light red dot, indicating it doesn't need attention, as shown in Figure 13. This could be realized using existing Jenkins plug-ins and changing them. The Radiator View¹⁰ seems like a good basis. An alternative could be to implement already existing software like Dashing¹¹, which provides the tools to generate dashboards very easily. Instead of showing the weather as proposed, color codes may be used for the tile backgrounds, either representing the status of the last build or the status of the past builds. A combination is also possible where the color represents the status of the last build and an opaque weather report image as the background. An example is given in Figure 15:

Website Build #3576 Merge pull request #813 from afterbranch Billy Last updated at 20:40	API Build #29 Profiling added Chavez Last updated at 20:40	Lefty Build #26 Version bump Chavez Last updated at 20:40	GoGeo Build #18 Version bump Chavez Last updated at 20:40
Iris Build #7 Fix readme Johnny Last updated at 20:40	Ruby Gem Build #1 Move over hidden files. Chavez Last updated at 20:40	Choices Build #4 Add circle.yml Chavez Last updated at 20:40	Project X Build #117 add new feature Mary Last updated at 20:40

Figure 15: Example dashboard for Jenkins.

The colors aren't too overwhelming and they're meaningful, since people are used to the colored dots in the Jenkins dashboard. A third, less tile-oriented dashboard is Team Dashboard¹². It also supports Jenkins plug-ins.

¹⁰<http://blog.matthewskelton.net/2013/03/11/what-makes-an-effective-build-and-deployment-radiator-screen/>

¹¹dashing.io

¹²http://fdietz.github.io/team_dashboard/

References

- [1] Stephen Few (2007). *Dashboard Design For Real-Time Situation Awareness*. [09/07/2015, published at: http://www.perceptualedge.com/articles/Whitepapers/Dashboard_Design.pdf]
- [2] Matthew Skelton. *What Makes an Effective Build and Deployment Radiator Screen?*. [09/07/2015, published at: <http://blog.matthewskelton.net/2013/03/11/what-makes-an-effective-build-and-deployment-radiator-screen/>]
- [3] Nick Smith. *Designing and Building Great Dashboards - 6 Golden Rules to Successful Dashboard Design*. [09/07/2015, published at: <https://www.geckoboard.com/blog/building-great-dashboards-6-golden-rules-to-successful-dashboard-design/>]
- [4] Zach Gemignani. *A Dashboard Alerts Checklist*. [09/07/2015, published at: <http://www.juiceanalytics.com/writing/dashboard-alerts-checklist/>]
- [5] No author specified. *A Guide to Creating Dashboards People Love to Use*. [10/07/2015, published at: http://www.cpoc.org/assets/Data/guide_to_dashboard_design1.pdf]
Images:
 - [6] *Sun*. [<http://azcoloring.com/coloring-page/112445>]
 - [7] *Rain cloud*. [<http://www.clipartpanda.com/categories/animated-rain-clouds>]
 - [8] *Red dot*. [<http://www.thepointless.com/reddot>]
 - [9] *Pink dot*. [<http://www.create-a-mural.com/dry-erase-11-soft-pink-dot-decal.html>]
 - [10] *Example dashboard*. [<https://camo.githubusercontent.com/1509a366bd27fad24ba09c45793308a85bcbf8bf/687474703a2f2f662e636c2e6c792f6974656d732f336330743076335a31313145306f3436333033322f53637265656e25323053686f74253230323031332d31302d30332532306174253230382e34362e3437253230504d2e706e67>]

Appendix B Enquête 1

Enquête De enquête bestond uit een reeks voorbeeldtegels met bepaalde kenmerken (zie Figuur 16) die ook in de enquête zelf uitgelegd werden (het deel boven “Preferences”). De enquête bestond uit volgende vragen, met ruimte om commentaar te voorzien:

“Hello” == name of build

1. No progress meter for the currently running build. Background color indicating the success/failure of the last completed build, big weather report about the last builds.
2. Small meter showing the progress of the currently running build, weather report at the bottom, background color indicating success/failure of last completed build.
3. Small meter with weather report on top, background color indicating success/failure.
4. Circular meter with weather report on top, background color indicating success/failure of last build.

Preferences (circle the preference)

- Meter/no meter; in case of meter:
 - Large/small
 - On top/at bottom
 - Circular/dashboard-like
- Weather report/no weather report
- Background color/no background color (colored dot is better)

Resultaten De resultaten zijn te zien in Tabel 1. Let op, niet iedereen heeft op elk onderdeel geantwoord, daarom zijn er niet op elke vraag 9 antwoorden.

	Yes	No
Meter	8	0
-Large	4	4
-On top	1	6
-Circular	3	5
Weather report	8	1
Background color	9	0

Table 1: Resultaten van de eerste enquête

Bijkomende opmerkingen zijn de volgende:

- “Would combine 1 and 3. Big weather report with small meter at the bottom”
- “It would be nice if we can show the failure count for failure builds”
- “I would replace ‘last updated’ with latest finished build and building since (if in progress). Only show meter when busy building”

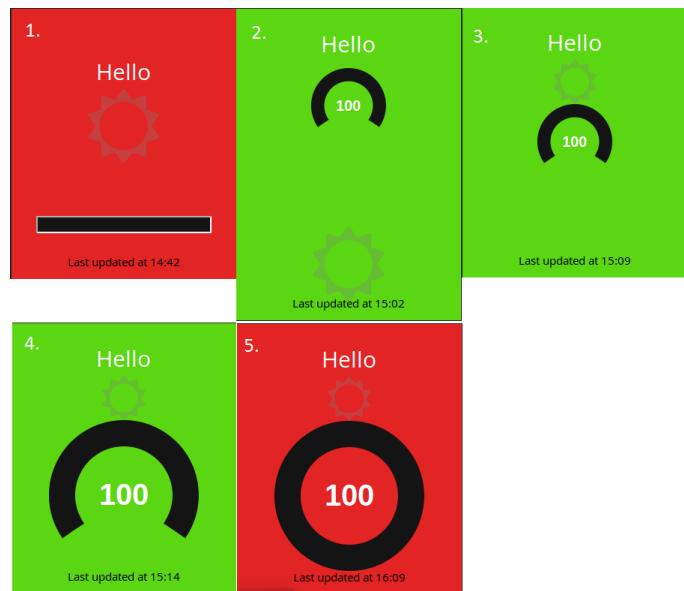


Figure 16: Voorbeeldwidgets.

- “Overview of last builds = list or circle with % of failed/success”
- “I’d put more visibility for the weather report/more important than current build progress. Maybe also add the date of the last build/number of test or failing tests?”

Appendix C Enquête 2

Enquête Er werd een formulier rondgegeven waarop men positieve en negatieve puntjes kon opschrijven, alsook suggesties van wat men nog graag zou zien op het dashboard.

Resultaten De positieve opmerkingen die gegeven zijn, zijn de volgende:

- “The requirements are met.”
- “Good looking/shows clearly the status of the build.”
- “Looks good visually, nice colors, nice layout”
- “Very clear indication of which ‘project’ failed. → Big title + background color.”
- “Good look and feel. Realtime updates - very nice.”
- “3 colors → in 1 quick view you see the complete overview.”

Suggesties en negatieve puntjes die aangehaald zijn, zijn de volgende:

- “Build name is in case of gray or yellow background not readable.”
- “It does not show all the jobs. It shows maybe too much info (instead of the progress bar, you could make the background blink when a build is in progress). ⇒ It would be better to have different ‘zones’. The idea would be to have jobs separated by version and some jobs ‘highlighted’. What matters is that when there is a release soon, we can check visually the status of all the jobs related to that release. There is no need to have all the weather info/progress for all the builds.”
- “Build progress is less important so it can be shown less big.”
- “Add an RSS feed for news items.”
- “Wishlist
 - Clicking a box should take us to the job. Top right → add link to Jenkins home, maybe the list of all jobs (the Jenkins server) - low priority.
 - Time in AM/PM.
 - The colours during ‘build in progress’ and ‘build disabled’ are not the same as in Jenkins.
 - Toggle button which can group per product and per release (low priority).
 - Questions:
 - * Will this survive a Jenkins upgrade?
 - * How easy is it to add a new build indicator?”

Het is duidelijk dat enkele mensen nog vragen hadden bij het dashboard, dus is hen nog extra uitleg gegeven. Enkele van de voorstellen die gedaan zijn, zijn nog verder uitgewerkt.

Appendix D Bedrijfservaring: competenties

- Technische competenties die ik hoop te verbeteren:

Bijleren over version control: mijn taak zal erin bestaan de werking van het version control systeem overzichtelijker te maken. Op deze manier zal ik een inzicht krijgen in hoe version control gebruikt wordt in een bedrijfsomgeving.

Leren werken met Jenkins: ik had nog nooit van dit continuous integration systeem gehoord, maar het lijkt me nuttig en interessant om hierover te leren. Die ervaring kan ik eventueel gebruiken in volgende opdrachten.

Mij inwerken in een reeds lopend project: aan de KU Leuven moeten we meestal projecten vanaf nul beginnen maken, hier zal ik in een reeds bestaand project terechtkomen. Me hierin inwerken en leren hoe andere mensen de zaken aanpakken, lijkt me heel leerrijk.

- Persoonlijke competenties die ik hoop te verbeteren:

Leren werken in grote teams: aan de KU Leuven werk je ten hoogste met 6 mensen samen, wat al een uitdaging vormt op zich. Nu zal ik in een bedrijfsomgeving komen waarbij men met veel meer mensen samenwerkt, hieruit verwacht ik dus te leren hoe je moet samenwerken in heel grote groepen.

Plannen op lange termijn: bij projecten aan de KU Leuven kreeg je interne (kleine) deelopdrachten bij projecten, hier zal ik meer zelf moeten plannen en zien wat realiseerbaar is binnen vooropgestelde tijd.

referentielijst?