AFSTUDEERVERSLAG

Ontwikkelen van een gebruikersvriendelijk stuk software voor het maken en aanpassen van algoritmes bij NewCompliance.

Naam: Menno Postma
Studentnummer: 09013342
Bedrijf: NewCompliance
Studie: Informatica
School: De Haagse Hogeschool
Stagebegeleider: A.A.A.M. Jacobs
Bedrijfsbegeleider: M. Koene

Datum: 24-3-2016

Plaats: Zoetermeer

Voorwoord

Beste lezer,

Dit verslag is geschreven in het kader van het afstuderen voor de opleiding informatica aan de Haagse Hogeschool. Het is bedoeld om de lezer inzicht te geven in de werkzaamheden die ik heb verricht tijdens mijn afstudeerstage. Het document is gericht aan personen met ICT kennis en kan dus vakspecifieke jargon bevatten.

-Menno Postma

Inleiding

In dit document zal ik de werkzaamheden tijdens mijn afstuderen bespreken. Ik zal daartoe beginnen met de achtergrond informatie. In dit hoofdstuk vertel ik wat over het bedrijf waar ik de afstudeeropdracht heb uitgevoerd. Vervolgens ga ik in op de probleemstelling voor het project.

In het hoofdstuk plan van aanpak leg ik het plan uit voor het verloop van het project. Hier maak ik onder andere de keuze voor de softwareontwikkelmethode. De rest van het document verteld chronologisch de werkzaamheden tijdens het project. Het laatste hoofstuk bestaat uit zelfreflectie. Ik ga in dit hoofdstuk mijn eigen proces en opgeleverde producten beoordelen. In dit hoofdstuk bespreek ik ook de leerdoelen die ik voor de stage heb gekozen en of ik deze leerdoelen naar mijn mening heb bereikt.

Inhoudsopgave

1 Achtergrondinformatie

1.1 bedrijf

De afstudeeropdracht wordt uitgevoerd bij NewCompliance. Een bedrijf dat zich richt op het verbeteren van patiëntveiligheid met het aanbieden van softwareoplossingen aan zorginstellingen.

Het idee voor het bedrijf is voortgekomen uit een afstudeer opdracht. De twee oprichters van het bedrijf, Bo Wiesman en Michael van Schie, hebben tijdens hun studie bedrijfskunde aan de Erasmus universiteit van Rotterdam het bedrijf NewCompliance opgezet.

Voor hun afstudeeropdracht in 2005 hadden ze als opdracht gekregen een product op de markt te brengen. Hun idee was om toilet papier op de markt te brengen waar berichten en reclame op stond. Met dit idee zijn ze vervolgens naar een toiletpapier producent gestapt. Het idee sloeg goed aan, maar de producent had zelf nog een toevoeging. Zo wilden ze weetjes en regels op het toiletpapier zetten dat voor ziekenhuizen bestemd was. Een van deze teksten die op het toiletpapier stond was als volgt: “Maar 45% van de dokters wast elke keer netjes hun handen”.

Hierna hebben ze onderzoek gedaan naar het zeep gebruik door dokters en ziekenhuis personeel. Uit dit onderzoek is gebleken dat het zeep gebruik soms wel met 35% toegenomen was. Hieruit was af te leiden dat de teksten op het toiletpapier er voor zorgden dat ziekenhuis personeel vaker hun handen wassen door de teksten op het toiletpapier. De teksten hebben er dus voor gezorgd dat de hygiëne in het ziekenhuis aanzienlijk toegenomen is. Uiteindelijk zijn ze niet verdergegaan met dit idee.

In het volgende jaar 2006 werd het bedrijf NewCompliance opgenomen in de YES!Delft. Dit is een incubator voor start-up bedrijven. YES!Delft maakt deel uit van de Technische Universiteit Delft. Het ondersteund en inspireert studenten in het opzetten en in de ontwikkeling van hun bedrijf. Uiteindelijk is het bedrijf officieel opgericht als een spin-off van TU Delft.

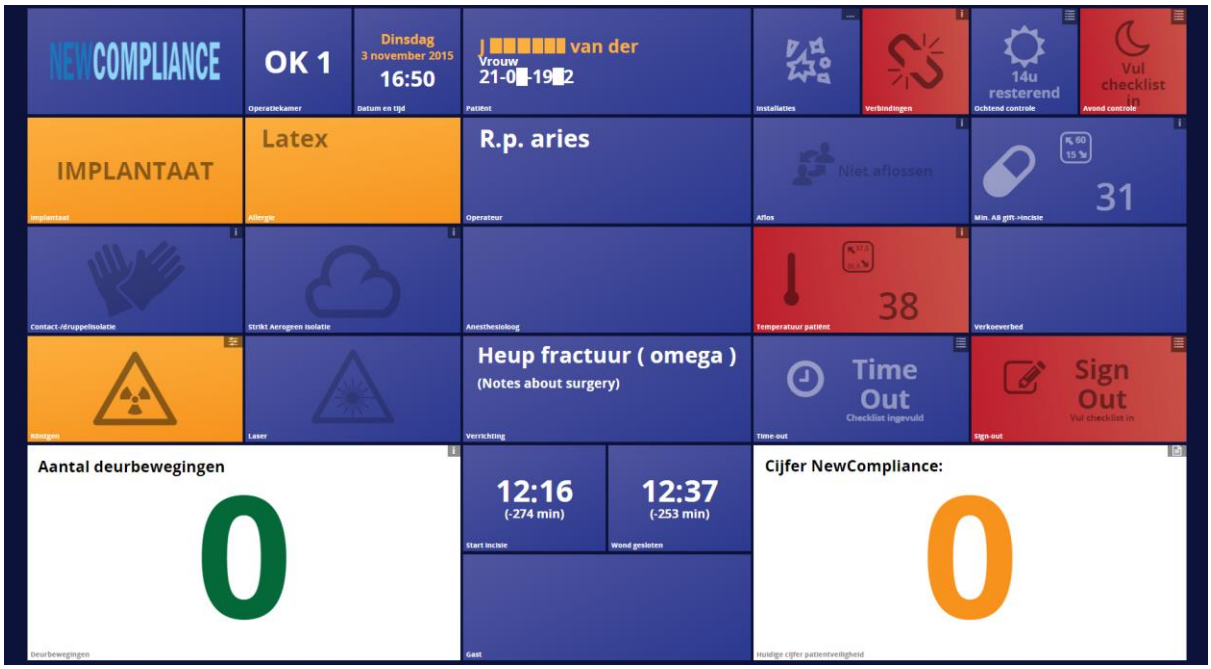
NewCompliance richt zich nu geheel op patiënt veiligheid binnen ziekhuizen. Ze hopen de patiëntveiligheid te verbeteren door middel van innovatieve en toekomstgerichte producten. De nadruk word hierbij gelegd op technologische oplossingen voor de problemen in de ziekenhuizen.

Het resultaat van deze inspanningen is dat NewCompliance, in 2010, als eerste en enige commerciële partij een eervolle vermelding voor de IGZ ZorgVeiligprijs 2010 hebben ontvangen. Dit is een prijs voor het beste initiatief op het gebied van patiënt- en cliëntveiligheid in de zorg.

Vervolgens is NewCompliance begonnen met het ontwikkelen van een deurteller- systeem. Dit systeem houd bij hoe vaak de operatiekamer deur open gaat gedurende de operatie. Deze informatie kan dan teruggekoppeld worden aan het ziekenhuis zodat ze inzicht krijgen in hoe vaak de deur gebruikt word. Dit systeem is uiteindelijk uitgegroeid tot het OperatieKamer Monitor Systeem. Dit systeem bied meer mogelijkheden om verschillende parameters bij te houden rond de operatiekamer. Een van de belangrijkste punten was het bijhouden van gebeurtenissen die betrekken hebben tot post operatieve wond infecties, ook wel POWI’s genoemd.

Deze OK cockpit is het belangrijkste product van de IT afdeling van NewCompliance. De parameters worden weergeven op het dashboard. Het dashboard is opgebouwd uit meerdere rechthoekige tegels. Op elke tegel word een module weergeven. Een module kan informatie over de patiënt of operatie weergeven maar kan ook functionaliteiten hebben. Het ‘unique selling point’ van de OK cockpit is dat het de enige monitoring software voor ziekenhuizen is met de focus op het visueel

weergeven van informatie. Er wordt gestreeft naar zo min mogelijk tekst op het dashboard. Dit wordt onder andere gedaan door veel gebruik te maken van iconen en kleuren.



AFBEELDING 1: SCREENSHOT VAN EEN DASHBOARD VAN DE OK COCKPIT

In juni 2012 is NewCompliance een distributeur van Vernacare geworden. Hiermee zijn zij de enige distributeur van Vernacare in Nederland. Vernacare verzorgt een systeem dat een alternatief vormt voor de huidige roestvrijstalen ondersteken en plastic urinalen. Hiervoor maken zij pulpproducten die weggegooid kunnen worden na gebruik. Hiervoor worden ze in speciale pulp disposal units gegooid. Door deze verandering was er een gebruik aan ruimte op de huidige werkplek. Hierom zijn ze in 2012 naar een nieuwe kantoor in Zoetermeer verhuisd. Op deze nieuwe locatie was genoeg ruimte voor de opslag van de pulpproducten.

In 2014 is de OK Cockpit van NewCompliance als finalist geëindigd voor de IGZ ZorgVeiligprijs 2014. Uit de nominatie blijkt dat de patiëntveiligheid die ontstaat door het gebruik van de OK Cockpit niet onopgemerkt is gebleven bij de IGZ. Deze erkenning is een bevestiging dat NewCompliance op de goede weg is om de zorg in Nederland veiliger te maken.

Voor de softwareontwikkeling hanteert NewCompliance de softwareontwikkelmethode SCRUM. Dit is al snel terug te zien door de beperkte documentatie. Er wordt voorkeur gegeven aan persoonlijk contact boven documentatie. De sfeer op de werkvloer is erg informeel, er is altijd plaats voor een grapje. Er wordt binnen het team nauw samengewerkt. Er wordt vaker even kort overleg gehouden dan dat er een lange bespreking wordt ingepland. Ook wordt er regelmatig aan ‘pair programming’ gedaan. Bij deze manier van programmeren waarbij twee developers tegelijk op hetzelfde scherm met hetzelfde stuk code bezig zijn wordt de geschreven direct gereviewed. Voor complexe stukken code is dit een groot voordeel.

Tijdens mijn afstuderen zal ik meewerken aan een nieuwe versie van de OK cockpit. Deze nieuwe versie gaat ingezet worden in de Intensive care van meerdere ziekenhuizen. Ik zal werkzaam zijn als software developer in de IT afdeling van NewCompliance. Hier zijn op dit moment 6 fulltime medewerkers en 1 parttime medewerker werkzaam.

1.2 aanleiding

Binnen NewCompliance is sinds kort het Thalea project gestart. Dit is een Europees project die in het leven is geroepen om een IT-oplossing te vinden om het herstel van patiënten op de intensive care (IC) te versnellen, zodat de patiënten sneller kunnen worden ontslagen. Gevraagd is een platform waar patiënten worden gemonitord op verschillende parameters. Medisch personeel moet hiermee snel de status van een patiënt kunnen beoordelen. Het systeem moet meerdere ziekenhuizen aan elkaar koppelen zodat (anonieme) informatie kan worden uitgewisseld. Met deze informatie kan het systeem mogelijk diagnoses voorstellen en behandelingen suggereren.

NewCompliance wil aan deze eisen voldoen met een nieuw systeem genaamd IC cockpit. Het bestaande OK cockpit systeem zal de basis vormen en wordt omgebouwd en uitgebreid tot de IC cockpit.

1.3 probleembeschrijving

Wanneer een alarm weergeven moet worden en hoe het rapport wordt opgebouwd in het huidige OK cockpit systeem, wordt bepaald door complexe algoritmes. Het is lastig te voorspellen welke algoritmes door de klant gewenst zijn. Als een klant na de oplevering van het systeem niet tevreden is met de huidige algoritmes is het de taak van NewCompliance om de nieuwe algoritmes te bouwen en implementeren. Dit maakt de klanten sterk afhankelijk van de leverancier. Daarnaast kost het veel tijd voordat de gewenste veranderingen aan de algoritmes daadwerkelijk bij de klant geïmplementeerd zijn.

De IT-oplossing voor het Thalea project moet zo onafhankelijk mogelijk van de leverancier zijn. Daarom is een eis vanuit Thalea een ‘algoritme systeem’ waarmee de gebruiker zelf algoritmes kan aanmaken en wijzigen.

1.4 doelstelling

Dit probleem vraagt om systeemdeel waarmee een persoon zonder IT kennis zelf veranderingen kan maken aan de algoritmes. De interface moet het bouwen en/of aanpassen van algoritmes op een gebruiksvriendelijke wijze mogelijk maken.

Met een dergelijk systeem krijgt de gebruiker uiteraard erg veel vrijheid om de software aan te passen. Dit kan leiden tot fouten of grote ongewenste veranderingen aan het systeem. Om dit te voorkomen moeten beperkingen worden ingebouwd.

Wanneer dit systeemdeel gerealiseerd is zal de IC cockpit een belangrijke eis vanuit Thalea ondersteunen en wordt de kans dat het project slaagt sterk vergroot.

2 Plan van aanpak

De eerste stap van dit project is het opstellen van het plan van aanpak. Hierin zijn belangrijke keuzes gemaakt voor het uitvoeren van het project.

2.1 Projectorganisatie

Het betreft een een-mans project. Dit betekend dat ik zelf alle rollen binnen het project zal vervullen. Daarnaast zijn er buiten het project een aantal rollen die door andere belanghebbende worden vervuld.

Sylvester Oosterbos: Softwarearchitect. De softwarearchitect overziet de bouw van het gehele Thalea systeem waarvan het algoritme-systeem dat tijdens dit project gebouwd zal worden slechts een onderdeel is. Keuzes voor de technische aanpak die invloed hebben op het gehele systeem moeten besproken worden met de softwarearchitect. Voor technische vragen kan tevens de softwarearchitect worden aangesproken. Sylvester zal wekelijks de door mij gemaakte code reviewen. Vooral om te controleren of ik mij aan de in NewCompliance vastgestelde code conventie voldoe.

Guillaume van Lamsweerde: Softwareanalist. De softwareanalist heeft contact met de eindgebruiker en zal zo gedetailleerd mogelijk de eisen en wensen achterhalen. De Thalea wijde requirements zullen aangeleverd worden door de softwareanalist. Deze zal ik vervolgens zelf moeten verfijnen tot meer gedetailleerde requirements. Functionele keuzes moeten besproken worden met de softwareanalist. Daarnaast bied de softwareanalist suggesties voor front-end design.

Bo Wiesman: Product-owner. De product-owner is eind-verantwoordelijk voor het product. De product owner zal altijd het laatste woord hebben over het op te leveren product.

2.2 Softwareontwikkelmethode

Om het project in goede banen te laten verlopen is het verstandig een softwareontwikkelmethode te kiezen. Een softwareontwikkelmethode bied ondersteuning door tips, sjablonen en een globale planning aan te leveren. Er zijn een boel verschillende methoden die allemaal hun voor- en nadelen hebben. Welke methode het beste is, is sterk afhankelijk van het project.

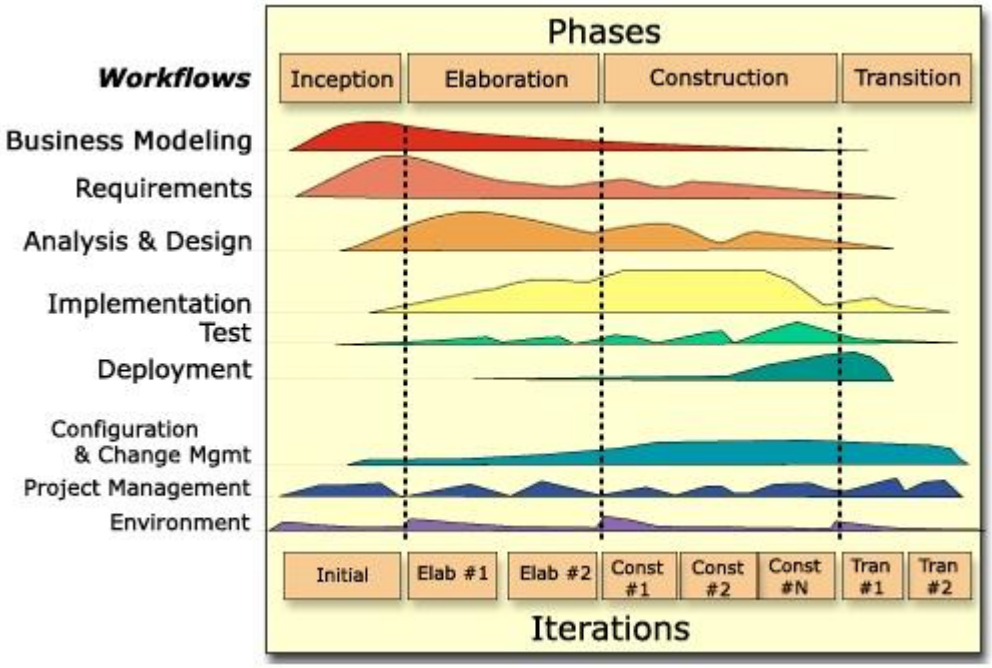
In ons project is het doel al vrij duidelijk. Dat komt doordat er vanuit Thalea eisen en wensen zijn aangeleverd. Hoe het doel bereikt moet worden is echter nog niet verder gespecificeerd. Ik verwacht dat er veel manieren zijn om het doel te bereiken. Tijdens het inlezen in de materie kwam ik er al gauw achter dat er veel manieren zijn om te programmeren zonder verdere IT-kennis. Daarnaast is het lastig voorspellen hoeveel tijd het ontwerpen en bouwen van het systeemdeel gaat kosten. Om deze reden is een iteratieve ontwikkelmethode gewenst. Een iteratieve ontwikkelmethode maakt het mogelijk om tussendoor het project bij te sturen. Anders dan bij een waterval methode, waarbij het gehele project vooruit wordt gepland, word bij een iteratieve methode het project in fases verdeeld en per fase een planning gemaakt.

twee veelgebruikte varianten van iteratief ontwikkelen zijn RUP en Scrum. Ondanks dat het beide een iteratieve ontwikkelmethode betreft verschillen ze sterk van elkaar. Scrum is een agile ontwikkelmethode en houd zich aan de principes die beschreven staan in de manifest voor agile software ontwikkeling:

- Mensen en hun onderlinge interactie boven processen en hulpmiddelen
- Werkende software boven allesomvattende documentatie
- Samenwerking met de klant boven contractonderhandelingen
- Inspelen op verandering boven het volgen van een plan

De planning van een project volgens scrum bestaat uit een 'product backlog'. Een lijst die gevuld wordt met alle taken die voor het project uitgevoerd moeten worden. Taken krijgen een prioriteit toegekend en de lijst wordt hierop gesorteerd. Een project bestaat uit een aantal sprints. Een sprint is een periode van 2 a 3 weken waar taken voor worden ingepland. Taken uit de product backlog met de hoogste prioriteit worden zo snel mogelijk ingepland.

Bij RUP of Rational Unified Process wordt het project opgedeeld in 4 fases en de taken opgedeeld in 9 disciplines. De intensiteit waarmee een bepaalde discipline wordt uitgevoerd is vooral afhankelijk van de fase waarin het project zich bevindt. In het schema van afbeelding 2 is per fase te zien in welke mate de 9 disciplines aan bod komen.



AFBEELDING 2: PROJECTVERLOOP VOLGENS RUP. OP DE HORIZONTALE AS STAAN DE FASES EN ITERATIES. OP DE VERTICALE AS STAAN DE DISCIPLINES.

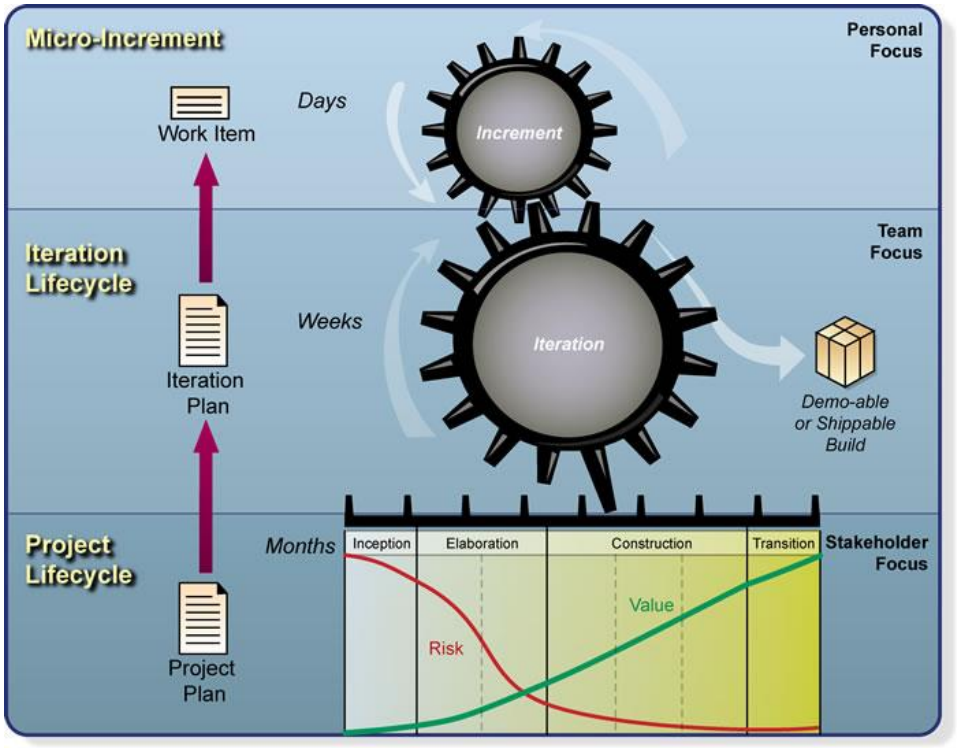
Zoals onderaan in afbeelding 2 is te zien zijn de elaboration fase, construction fase en transition fase verder onderverdeeld. Deze drie fases kunnen namelijk geïtereerd worden. Aan het eind van iedere iteratie moet een afgerond product worden opgeleverd.

RUP beschrijft per discipline welke documenten kunnen worden opgeleverd en in welke fase dit het meest relevant is. De hoeveelheid documentatie die door RUP wordt aanbevolen om tijdens het project te maken is veel. Zeker in een klein en kort project als die van ons zal dit veel tijd kosten die wellicht beter besteed kan worden aan de bouw.

openUP is een open source versie van RUP met een meer agile aanpak zoals Scrum. De hoeveelheid documentatie wordt vermindert doordat de methode stelt dat de documenten alleen gemaakt

moeten worden wanneer deze echt waarde toevoegen. Daarnaast wordt aangemoedigd de documenten zo kort en bondig mogelijk te maken en hoofdstukken en/of paragrafen die niet interessant zijn voor het project weg te laten. Daarnaast wordt net zoals bij SCRUM, directe communicatie belangrijker gevonden dan documentatie.

OpenUP introduceert verder de work-items list. Hierin staan alle taken van het project gesorteerd op prioriteit. Deze lijst kan gezien worden als de openUP equivalent van de product backlog van Scrum. Per iteratie worden work items uit de work items list ingepland. Elke afgeronde work item is een micro-increment die naar het afronden van de iteratie toewerkt. Elke iteratie werkt op zijn beurt steeds verder naar het afronden van het project. Dit proces is schematisch weergeven in afbeelding 3.



AFBEELDING 3: PROJECTVERLOOP VOLGENS OPENUP

Zo bied RUP het beste van twee werelden. De voordelen van agile ontwikkelen met de leidraad van RUP. Daarom is de keuze gemaakt om met softwareontwikkelmethode openUP te gaan werken.

2.3 Planning

De planning wordt gemaakt volgens openUP. Dit betekend dat het project van 17 weken opgedeeld zal worden in 4 fases met per fase een x aantal iteraties.

De inception fase bestaat uit kennismaking met het bestaande systeem en de gebruikte software en technieken. Er zal waarschijnlijk een kort onderzoek gedaan worden naar manieren van programmeren zonder IT-kennis. Ook zal er een begin gemaakt worden aan de requirements. Ik verwacht hier om en nabij 3 weken mee bezig te zijn.

De elaboration fase bestaat uit het vastleggen van de requirements en het maken van de bijhorende use cases. Vanuit deze requirements zal een functioneel design gemaakt worden. Naast het functioneel design wordt in deze fase ook een technisch design gemaakt. Ik heb gekozen voor een elaboration fase van 3 weken. Ik verwacht dat dit genoeg tijd bied om een eerste versie van de

designs op te leveren. Tijdens de construction fase kunnen de designs nog verder verfijnd worden. Meerdere iteraties lijken me dan ook niet nodig.

Tijdens de transition fase wordt het product gereedgemaakt om in productie genomen te worden bij de klant. De laatste tests worden uitgevoerd en de code en documentatie worden gecontroleerd. Indien nodig wordt er een handleiding gemaakt. Normaal gesproken wordt in deze fase het product daadwerkelijk in productie genomen en wordt er training gegeven aan de gebruikers. Dit valt tijdens dit project buiten de scope. 2 weken voor de transition fase zal daarom ruim voldoende moeten zijn.

De overgebleven 9 weken worden geplaatst in de construction fase. Tijdens de construction fase wordt het product gebouwd. Aan het eind van de fase moet het losstaande werkende product worden opgeleverd. Om het mogelijk te maken om tussendoor feedback te ontvangen en het project mogelijk bij te sturen heb ik de keuze gemaakt om de construction fase in 3 iteraties op te delen. Dit betekend dat elke 3 weken een (deel)product wordt opgeleverd waarover een demo gegeven wordt.

Zo komen we uit op de volgende grove planning voor het totaal van 17 weken:

- Inception fase (3 weken):
- Elaboration fase (3 weken):
- Construction fase1 (3 weken):
- Construction fase2 (3 weken):
- Construction fase3 (3 weken):
- Transition fase (2 week):

Naast de grove planning zal de work-items list worden opgesteld. In deze lijst staan alle taken die nog uitgevoerd kunnen worden tijdens het project; de zogenaamde work-items. Per iteratie of fase worden work-items uit de work-items list gekozen die uitgevoerd zullen worden tijdens de iteratie. De eerste versie van de work-items list zal tijdens de inception fase worden opgesteld.

2.4 Omgeving

Tools:

Tijdens het project worden een aantal tools gebruikt voor onder andere onderzoek en planning:

Trello werkt als een digitale versie van het whiteboard met post-it's. En zal gebruikt worden om de work-items list bij te werken en de planning per iteratie te maken. Deze tool zal je in dit verslag vaker terug zien tijdens het kopje 'planning' in elke iteratie.

Qippa is een tool die assisteert bij doen van een deskresearch. Voor een onderzoek zullen een groot aantal papers, boeken en websites gelezen worden. Qippa helpt daarbij door te onthouden wat wel of niet gelezen is en door suggesties te doen voor relevante stukken tekst. Tijdens het lezen kan aan belangrijke stukken tekst tags worden toegewezen. Met de tags kan later gemakkelijk het stuk tekst teruggevonden worden. Deze tool zal gebruikt worden tijdens het onderzoek.

Github is een versiebeheertool. Met github is je code veilig opgeslagen en is het mogelijk terug te keren naar een oudere versie van je gebouwde software.

Frameworks en API's:

Laravel is het php framework waar de server-side van de IC cockpit op gebouwd is. Laravel zorgt voor structuur in de code en implementeert het model-view-controller principe. Dit houd in dat de applicatie bestaat uit 3 lagen: model, view en controller. De model laag is verantwoordelijk voor de data in de database. In de model laag staat beschreven hoe de data eruit ziet. Daarnaast bied de model laag de functionaliteiten om te data aan te passen. Dataobjecten kunnen aangemaakt, gewijzigd en verwijderd worden met behulp van de model laag.

Als een http request wordt uitgevoerd wordt dit door Laravel afgevangen. In het routes bestand staat per url beschreven wat er gedaan moet worden. In de meeste gevallen wordt er gelinkt naar een controller methode in de controller laag.

De controller laag is verantwoordelijk voor het behandelen van events. Dit zijn doorgaans handelingen van een gebruiker. De controller behandeld events door bijvoorbeeld een berekening uit te voeren of data op te vragen aan de model laag en deze te manipuleren. Als de controller hier vervolgens mee klaar is roept hij de juiste view aan en zend de data mee. De view is verantwoordelijk voor het weergeven van de data. De view is de laag die zichtbaar is voor de gebruiker en wordt geschreven in html, javascript en CSS. Via de view laag kan een nieuwe http request worden uitgevoerd waarmee de cirkel rond is.

AngularJS is een client-sided javascript framework. Het framework werkt door eerste de HTML-pagina te lezen. Staan in deze pagina zogenaamde 'directives' dan wordt hier een model aan gekoppeld die is gedefinieerd in javascript. Een model beschrijft het gedrag van de pagina. Tussen de view (de html pagina) en de model bestaat een datakoppeling in twee richtingen. Dit zorgt voor automatische synchronisatie tussen de model en de view. Dit verminderd de noodzaak om DOM-elementen te manipuleren waardoor de applicatie overzichtelijk blijft en het prestatievermogen verbeterd.

3 Inception fase

3.1 plan

Het plan voor deze fase is om kennis te maken met NewCompliance en zijn tools en software. Verder zal er deskresearch gedaan worden naar de mogelijkheden voor programmeren zonder IT-kennis. Daarnaast zal er begonnen worden met de work-items list en zal er een begin gemaakt worden met het opstellen van de requirements. Aan het eind van deze fase zal een planning gemaakt worden voor de volgende fase. Echter wordt, om het document overzichtelijk te houden, deze planning pas besproken in het hoofdstuk van de fase waarvoor de planning gemaakt is.

Op te leveren producten:

- Resultaten deskresearch
- work-items list
- iteration plan: elaboration fase

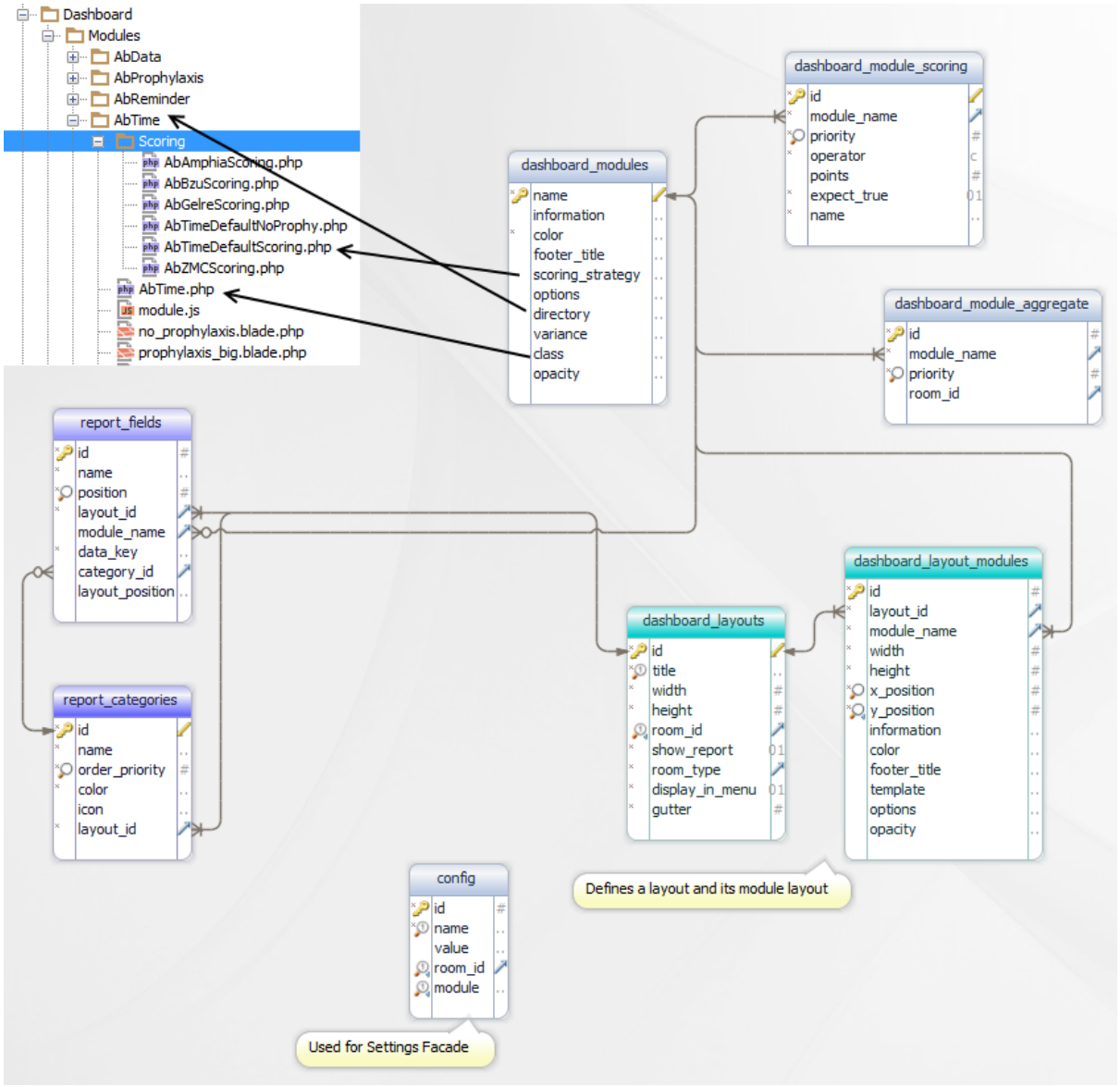
3.2 Uitvoering

3.2.1 Kennismaking OK-cockpit

De OK cockpit, de software waar een onderdeel voor gebouwd moet worden, is een webapplicatie. Als moedertaal is de software geschreven in php met het framework Laravel. Laravel zorgt voor de basisstructuur van de software.

Het dashboardsysteem is modulair opgebouwd. Een dashboard bestaat uit een of meer tegels. Elke tegel stelt een module voor en kan geheel onafhankelijk werken. In de beheertool kan voor een dashboard een lay-out worden gedefinieerd. Hier wordt vastgesteld hoe groot het grid is. Het grid stelt vast hoeveel tegels er mogelijk zijn in de hoogte en breedte. Vervolgens kan dit grid gevuld worden met de gewenste modules. Op deze manier kunnen snel nieuwe dashboards gebouwd worden voor de specifieke wensen van de klanten.

Om te kijken hoe dit op de achtergrond in zijn werkt gaat kijken we eerst naar de databasestructuur.



AFBEELDING 4: ERD VOOR DE STRUCTUUR VAN DASHBOARDS UIT DE BESTAAND DOCUMENTATIE VAN NEWCOMPLIANCE.

Een nieuwe module wordt aangemaakt door een nieuwe regel in de dashboard_modules tabel toe te voegen. De module wordt opgeslagen met een naam, informatie, standaard kleur en opacity. Maar belangrijker, de locatie en naam van de php klasse voor de module. Op deze klasse kom ik later terug.

De andere 2 tabellen waar we in geïnteresseerd zijn, zijn de dashboard_lay-outs en de dashboard_lay-out_modules. De dashboard_lay-out tabel slaat de verschillende lay-outs op. Een lay-out bestaat uit een width en height. Dit is de definitie van het grid, hier wordt dus aangegeven hoeveel tegels hoog en breed de lay-out wordt. In de lay-out wordt ook opgeslagen bij welke kamer deze hoort. In de dashboard_lay-out_modules worden de lay-out specifieke modules opgeslagen. In de lay-out module wordt opgeslagen van welk type module deze is, d.m.v. een vreemde sleutel naar dashboard_modules. Doormiddel van een vreemde sleutel naar dashboard_lay-outs wordt opgeslagen bij welk lay-out de module hoort. Op welke plek de module staat binnen de lay-out wordt bepaald door de x en y position attributen.

We gaan nu kijken wat er gebeurt op het moment dat er een lay-out is gedefinieerd en er wordt naar de pagina van het dashboard genavigeerd. Dit zou een globaal idee moeten geven van de werking van het dashboard van de OK cockpit.

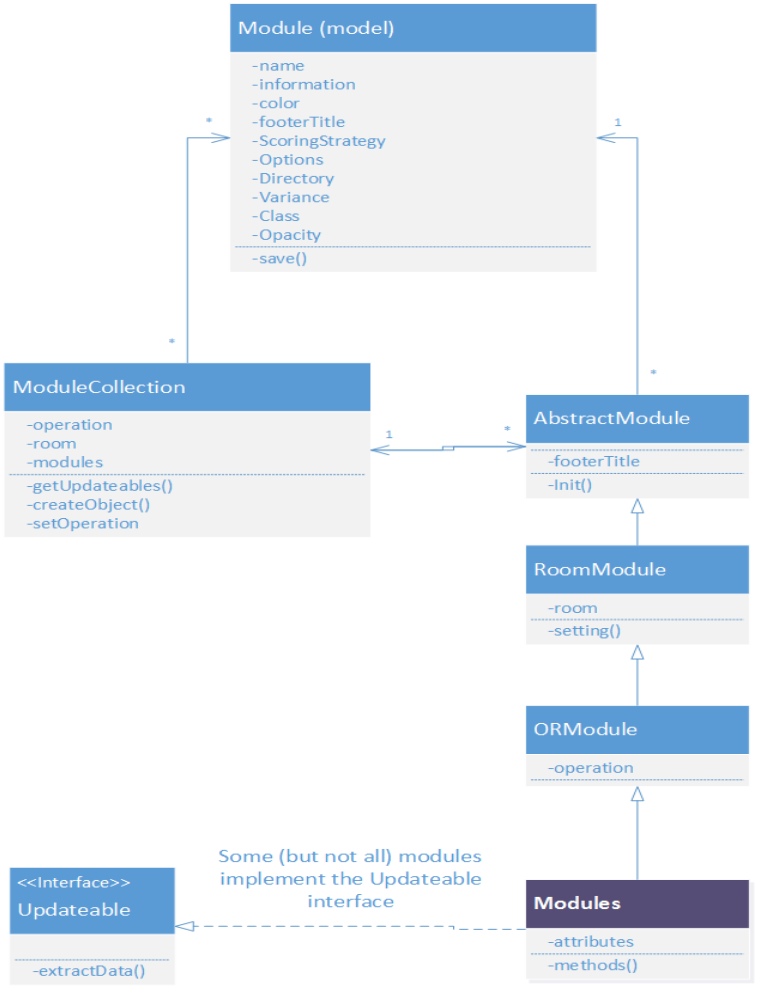
De gebruiker navigeert naar de pagina {*baseUrl*}/dashboard/room/1. Deze url wordt door het routes bestand doorgelinkt naar de methode showRoom in de DashboardController.

```
Route::get('/dashboard/room/{id}/{lay-out?}/{touch?}','DashboardController@showRoom
```

De regel in het routes bestand die url afvangt en doorlinkt naar de juiste controller methode. De namen tussen de accolades in de url zijn variabel. De variabelen met een vraagteken zijn optioneel.

In de showRoom methode wordt de bijhorende lay-out opgehaald. De lay-out wordt samen met de room meegegeven aan de dashboard view. In de html code van de view is een Angular directive aangegeven. Door middel van deze directive wordt de rest van het dashboard opgebouwd. Dit gebeurd in het dashboard.js bestand, waar de directive is gedefinieerd. Met behulp van Packery, een kleine Javascript library wordt de lay-out met de tegels opgebouwd in de webpagina.

Op dit moment zijn de tegels leeg. Het invullen van de tegels gaat door de update/all url aan te roepen. In de controller die bij deze url hoort worden de modules van de lay-out uit de database opgehaald. De modules worden opgeslagen in een object van de klasse ModuleCollection. De ModuleCollection maakt voor elke module een object aan van de klasse waarvan de locatie en naam is opgeslagen in de module.



AFBEELDING 5: KLASSEDIAGRAM VAN DE MODULE COLLECTION

In de klasse van elke module staat de werking van de module beschreven. De module bouwt een view op aan de hand van de data en instellingen. De view wordt op de juiste tegel van het dashboard geplaatst. Dit gebeurt voor elke module in het dashboard waarna de gehele dashboard geladen is. De update/all url wordt elke keer aangeroepen wanneer er een parameter die gebruikt wordt door het dashboard verandert in de database. Zo blijft het dashboard up-to-date en wordt zodoende de real-time informatie weergegeven.

3.2.2 Deskresearch

De volgende stap is het houden van een kort onderzoek. Het onderzoek bestond enkel uit deskresearch. Ik heb een groot aantal papers gelezen en van daaruit een conclusie getrokken en een keuze gemaakt voor de beste oplossing voor de implementatie.

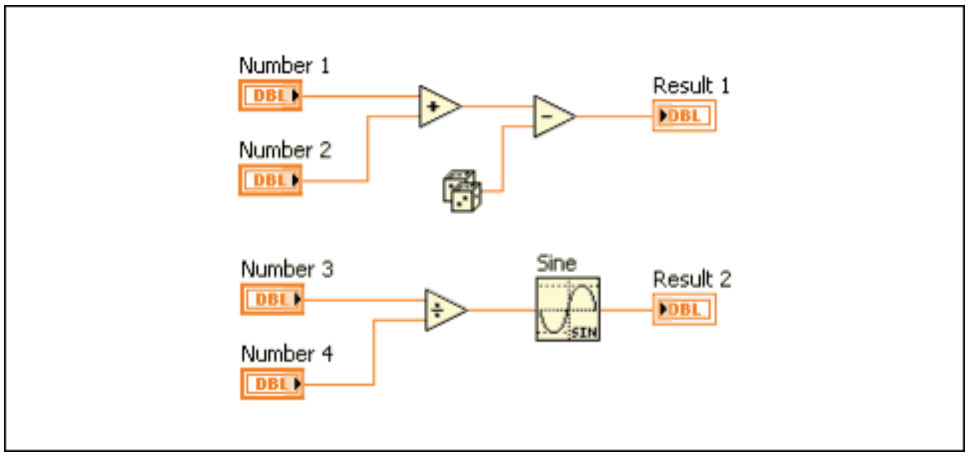
Er zijn veel manieren van programmeren. De meest bekende en meest gebruikte manier is het tekstueel programmeren. De meeste professionele programmeurs geven de voorkeur aan tekstueel programmeren. Het geeft de programmeur alle vrijheid en het is snel. Daarnaast kan er met tekstueel programmeren een boel informatie op een beperkt oppervlak weergeven worden. (Green, 1995)

Een nadeel van tekstueel programmeren is de syntax. Er is geen ruimte voor syntactische fouten. Dit maakt de drempel om te starten met programmeren groot. (Soloway, Ehrlich, Bonar, & Greenspan, 1984) Een visual programming language (VPL), is wellicht een betere oplossing voor non-programmeurs. Algoritmes worden gebouwd door blokken en/of vormen aan elkaar te plakken of te verbinden met lijnen. Met een VPL worden de eerder genoemde syntax problemen weggenomen. Stel je probeert een sequentie van operaties te bouwen dat syntactisch niet mogelijk is. Dit zal in een VPL niet mogelijk zijn doordat er in de programmeertaal restricties zijn ingebouwd om syntactische regels te forceren. In een tekstueel programma zal de gebruiker pas op de hoogte gebracht worden van de syntactische fout wanneer het programma wordt uitgevoerd.

Over de vraag of visuele programma's ook makkelijker leesbaar zijn, zijn meningen verdeeld. Onderzoek van green toont aan dat een visuele notatie consistent slechter presteert dan een tekstuele notatie in leesbaarheid. (Green, 1995) Pandey en burnet toonden echter het tegenovergestelde aan en beweerden dat visuele programma's beter leesbaar zijn. (Pandey & Burnett, 1993) Aangezien in deze onderzoeken verschillende algoritmes worden gebruikt kan geconcludeerd worden dat de beste notatie afhankelijk is van het algoritme dat ermee wordt weergeven. Deze stelling word ondersteund door green die stelt dat "Every notation highlights some kinds of information at the expense of obscuring other kinds. Not everything can be highlighted at once". (Green & Petre, Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework, 1996) Daarnaast is uit de onderzoeksresultaten te concluderen dat programmeurs het vaak prettiger vinden om met tekstuele dan visuele programmeertalen te werken. Kinderen en volwassenen zonder IT kennis vinden het doorgaans prettiger om met de visuele programma’s te werken. Het is dus van belang om te kijken wat de doelgroep is en welke informatie het meest relevant is voor de doelgroep en daar de keuze voor de programmeertaal op te baseren.

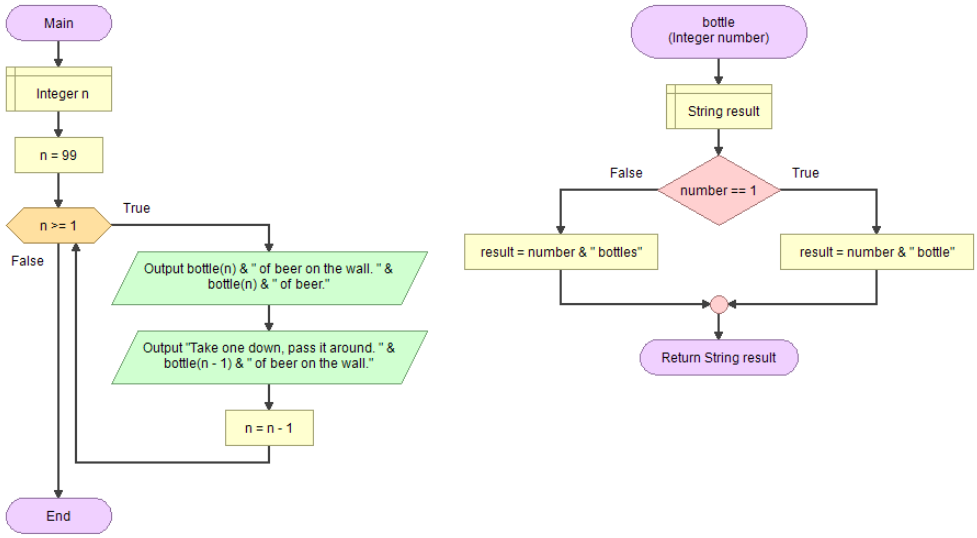
Er zijn 3 categorieën waar VPL's in zijn onder te verdelen. Hieronder ga ik ze bespreken:

- **Data flow**
  Data flow is een methode om algoritmes visueel te weergeven. Het algoritme wordt gevisualiseerd als een stroomschema waarbij er data door de pijlen stroomt. Er wordt geprogrammeerd door blokken in een canvas te verplaatsen en te verbinden met pijlen. Een blok is een operatie die met een of meerdere inputs een output genereert. Bijvoorbeeld een vermenigvuldig blok, waarbij een simpele rekensom wordt uitgevoerd en de uitkomst als output wordt gegeven. Hieronder een voorbeeld:



AFBEELDING 6: VOORBEELD VAN EEN ALGORITME IN LABVIEW. EEN DATA FLOW PROGRAMMEERTAAL.

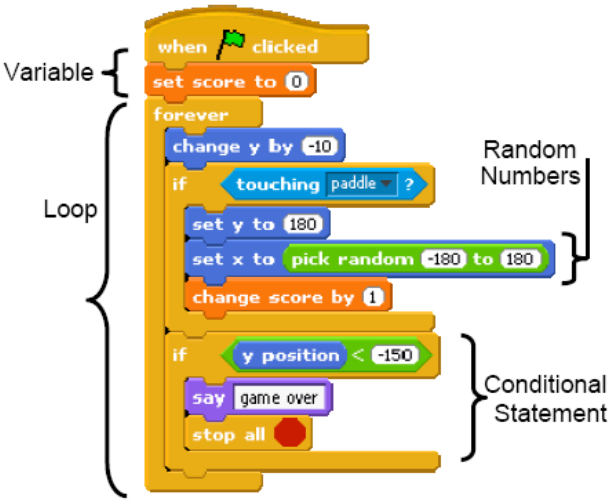
- **Control flow**
  De control flow visualisatie lijkt op die van de data flow. Echter bevatten de pijlen hier geen data maar stellen ze keuzes voor. Dit betekend dat de blokken alleen vergelijkingen en logische poorten kunnen bevatten zoals: is gelijk aan; is groter dan; input 1 en 2 zijn beide waar. Hieronder een voorbeeld:



AFBEELDING 7: VOORBEELD VAN EEN ALGORITME IN FLOWGORITHM. EEN CONTROL FLOW PROGRAMMEERTAAL.

- **Drag and drop**

Programmeertalen binnen de drag and drop categorie houden grotendeels de structuur aan van tekstuele programmeertalen. Echter wordt er in plaats van code getypt, bestaande blokken in een canvas gesleept. Deze blokken kunnen vervolgens als puzzelstukken aan elkaar geplakt worden om een algoritme te schrijven. Hieronder een voorbeeld:

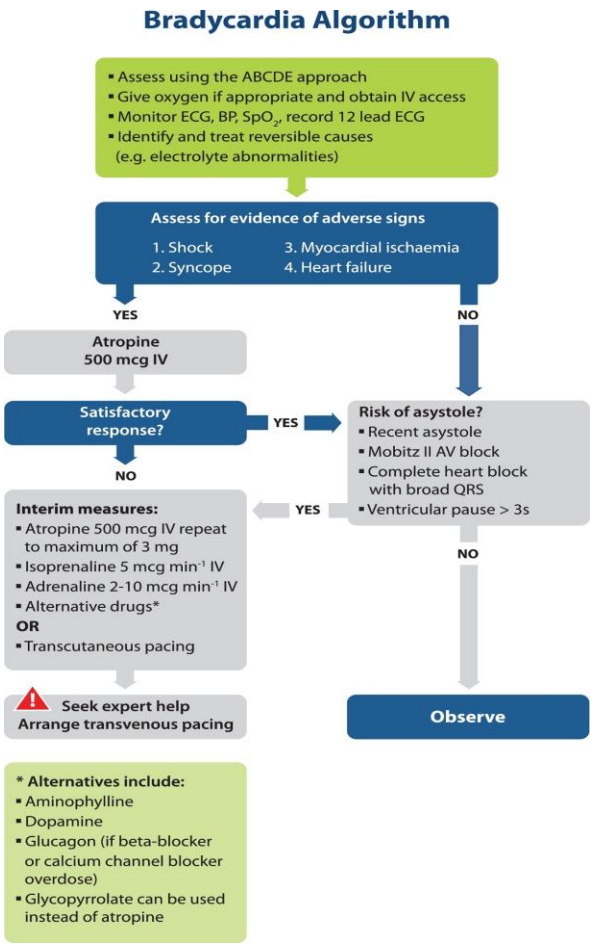


AFBEELDING 8: EEN VOORBEELD VAN EEN ALGORITME IN SCRATCH. EEN DRAG AND DROP PROGRAMMEERTAAL

Als we gaan kijken naar de doelgroep dan ontdekken we dat er in de medische wereld veel gebruik gemaakt word van stroomdiagrammen. Met name in het stellen van diagnoses en de behandeling van een patiënt worden zogenoemde medische algoritmes gebruikt.

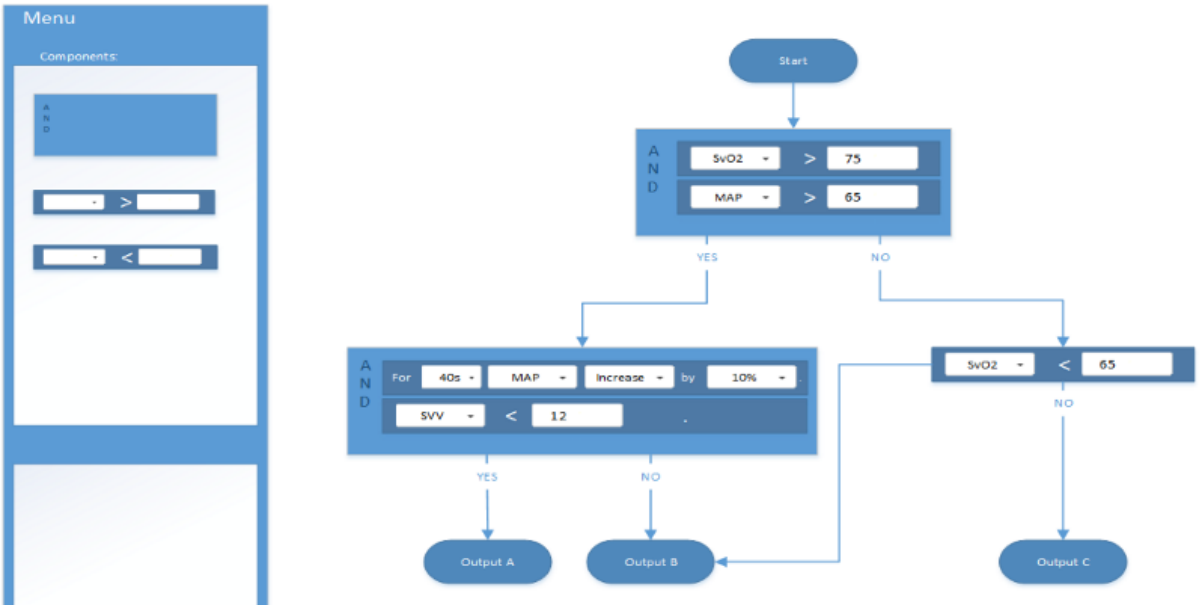
In het voorbeeld van een medisch algoritme in afbeelding 7 zien we dat het gaat om een control flow diagram; De blokken zijn vragen en de pijlen stellen keuzes voor. In een blok kunnen meerdere condities staan die bepalen of de uitkomst waar of niet waar is. Verder zien we dat de volgorde zo veel mogelijk van links boven naar rechtsonder gaat. Deze weergave kan toegepast worden als visuele programmeertaal. Dit kan mogelijk gemaakt worden door een stroomschema met drag en drop te combineren. Binnen een blok van het stroomschema worden de condities opgebouwd door middel van drag and drop. Op deze manier zou het mogelijk zijn om een werkend algoritme zoals in afbeelding 7 na te bouwen zonder dat de gebruiker syntactische kennis nodig heeft. In afbeelding 8 is te zien hoe dit er mogelijk uit ziet.

Als het algoritmesysteem zo gemaakt wordt dat de algoritmes zo veel mogelijk op medische algoritmes te lijken dan heeft de doelgroep waarschijnlijk weinig moeite om met de software om te gaan. Een groot nadeel is echter dat deze notatie erg veel ruimte in beslag neemt waardoor grote complexe algoritmes mogelijk erg onoverzichtelijk en onleesbaar worden. Een ander bijkomend nadeel is dat er geen open-source software bestaat voor control flow programmeren. Dit zou betekenen dat het volledige algoritmesysteem van begin af aan gebouwd moet worden. Iets wat uiteraard erg veel tijd zal kosten. De deadline voor het eerste prototype van de IC cockpit is begin mei. Met dit prototype moet de organisatie van het Thalea project overtuigd worden dat onze oplossing de beste is. We willen tegen die tijd een zo groot mogelijk deel van de requirements hebben geïmplementeerd. Het maken van een control flow algoritme systeem zal simpelweg te veel tijd kosten.



AFBEELDING 9: VOORBEELD VAN EEN MEDISCH ALGORITME

Een andere mogelijkheid is de dataflow weergave, maar deze valt al snel af. De dataflow weergave is namelijk slecht in het weergeven van volgorde. Aangezien medisch personeel juist gewend is aan stroomschema en stappenplannen, waarbij volgorde een cruciaal onderdeel is, willen we volgorde met onze weergave juist naar voren laten komen.



AFBEELDING 10: MOCKUP VOOR EEN PROGRAMMEERTAAL DIE CONTROL FLOW MET DRAG AND DROP COMBINEERT.

Dan blijft een weergave volledig gebaseerd op ‘drag and drop’ over. Deze weergave sluit waarschijnlijk minder goed aan op de doelgroep. Maar met een korte scholing moet dit geen probleem zijn verzekerd Guy van Lamsweerde, de softwareanalist van NewCompliance. Guy heeft veel contact met de doelgroep waardoor hij dit beter kan inschatten dan ik. Het grootste voordeel van drag en drop is dat deze weergave op dit moment razend populair is om leerlingen het programmeren aan te leren. Hierdoor zijn er meerdere voorbeelden om naar te kijken en inspiratie op te doen. Daarnaast bestaat er een open source web-API om een drag and drop programmeeromgeving te bouwen. Deze API heet Blockly en de eerste indruk van mij is zeer positief. Gezien de voordelen is keuze definitief; het algoritme systeem zal de drag and drop weergave hanteren en gebruik maken van de Blockly API.

3.3 review

De inception fase bestond uit het doen van een kort onderzoek en het bekend worden met de software en tools van NewCompliance. Over visueel programmeren en de voordelen daarvan was veel informatie te vinden. Blijkbaar is dit op dit moment een populair onderwerp. Dit heeft mogelijk te maken met de steeds grotere vraag naar programmeurs. Meerdere papers suggereerde dat de visuele programmeeromgeving goed moet aansluiten op de manier van denken van de doelgroep. Op basis hiervan heb ik ook onderzoek gedaan naar de doelgroep. Mijn eerste conclusie was op deze informatie gebaseerd. Een visuele programmeeromgeving geïnspireerd door control flow diagrammen met een stukje drag en drop om alle tekstuele syntax weg te nemen. Helaas zou deze oplossing te veel werk kosten en is gekozen voor een volledige drag and drop programmeeromgeving. Nu twijfel ik of de tijd die ik in het onderzoek heb gestoken het waard is geweest. De keuze is uiteindelijk gevallen op een programmeeromgeving die niet aansluit op de conclusie uit het onderzoek. Ik heb echter wel veel algemene kennis opgedaan over visuele programmeeromgevingen. Wellicht dat dit in de toekomst toch nog zijn waarde bewijst.

De OK cockpit is een uitgebreid stuk software. De modulaire opbouw maakt het complex en lastig voor een nieuwkomer om te achterhalen hoe de webpagina's worden opgebouwd. Ik denk dat ik na deze fase wel een goed idee heb van de werking van de software. Voldoende om een ontwerp te maken voor het te bouwen systeemdeel.

4 Elaboration fase

4.1 Plan

Het doel van deze fase is om alles gereed te hebben zodat in de volgende fase direct begonnen kan worden aan de constructie. Om dit de bereiken moeten eerst de eisen en wensen worden opgesteld. Vervolgens kunnen hiermee de use cases worden opgesteld. Daarna kan begonnen worden met het maken van het front-end en back-end design.

4.2 Uitvoering

Voordat ik kan beginnen met het ontwerpen van het systeemdeel moet ik duidelijk eisen en wensen hebben. Daarvoor heb ik een bespreking ingepland met de architect, analist en product owner. Voor de bespreking heb ik mijn eigen ideeën uitgewerkt. Mijn plan was om in de modulebeheer pagina, waar modules gewijzigd kunnen worden, een extra optie aan de modules te geven. Met deze optie kunnen regels worden ingesteld door de gebruiker. Een regel bestaat uit een trigger en een effect. Een effect manipuleert de visuele weergave van de module, zoals het veranderen van de kleur. Wanneer dit effect plaatsvind wordt bepaald door de trigger. De trigger is een algoritme dat gebouwd is met de algoritme tool. Deze tool zal onafhankelijk zijn van de OK cockpit. Een algoritme wordt met de tool gebouwd en opgeslagen in de database. Vervolgens is dit algoritme te selecteren in de OK cockpit als trigger. Op deze manier is de algoritme tool beter herbruikbaar zodat hij misschien ook in andere toepassingen gebruikt kan worden.

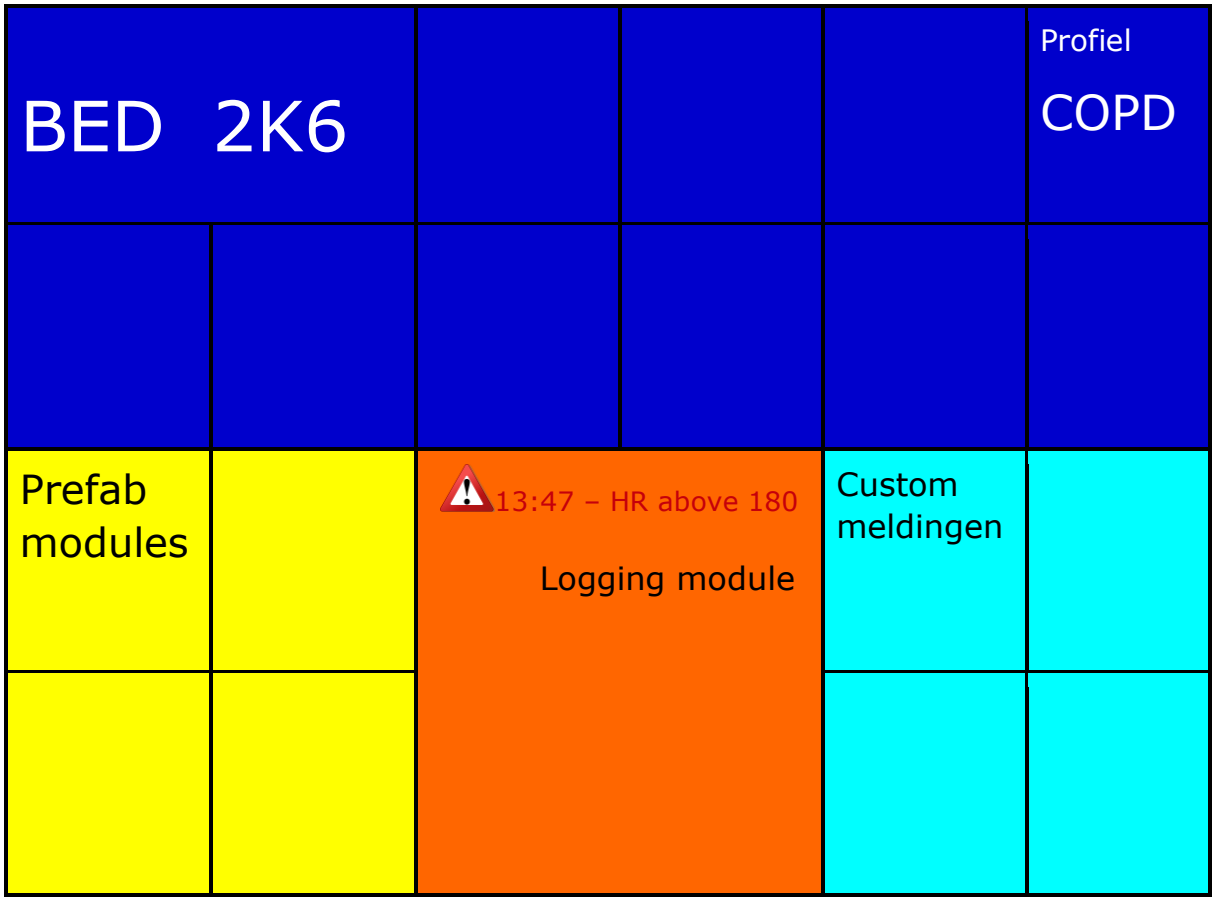
Deze oplossing zal waarschijnlijk wel ten koste gaan van de performance. Immers zal elke regel een eigen algoritme hebben. Dit terwijl een deel van de logica misschien wel hetzelfde is als een van de andere algoritmes. Dit kan opgelost worden door de effecten binnen de visuele programmeer omgeving te plaatsen. Er zal dan een zelfgemaakt blokje moeten komen waar een effect op geselecteerd kan worden. Dit blokje kan vervolgens in het algoritme opgenomen worden. Wordt tijdens de uitvoer van het algoritme het blok bereikt dan zal het desbetreffende effect plaatsvinden.

Met deze oplossing is het wel gewenst om de algoritme tool bereikbaar te maken vanuit de OK cockpit. De visuele manipulaties van de module zitten namelijk direct in het algoritme verborgen.

Tijdens de bespreking werd positief gereageerd op mijn plannen. Ik ben toen begonnen met de verdere uitwerking van dit idee. Echter kort daarop tijdens een algemene bespreking van het Thalea project zijn de plannen veranderd. Tijdens dit gesprek is de indeling van het dashboard van de nieuwe IC cockpit besproken. Er is al eerder door NewCompliance vastgesteld dat er 3 verschillende dashboards gebouwd zullen worden voor 3 verschillende niveaus. Het laagste niveau is het patiënt niveau. Dit is het dashboard dat naast een bed in de intensive care weergeven wordt. Op dit dashboard is informatie over de patiënt en zijn status te zien. Een niveau hoger is het ward niveau. Op dit dashboard zijn alle intensive care bedden van een afdeling zichtbaar. Dit dashboard heeft als primair doel het personeel te alarmeren als een patiënt dringend hulp nodig heeft. Het 3e en hoogste niveau is het centrale niveau. Op dit dashboard wordt informatie van het gehele ziekenhuis weergeven. Het doel van dit dashboard is om ziekenhuis wijde analyses uit te voeren.

We zijn toen het ontwerp van het dashboard op patiënt niveau gaan bespreken. Dit dashboard heeft de hoogste prioriteit omdat wordt verwacht dat dit dashboard te meeste waarde zal toevoegen aan het prototype dat eind mei wordt opgeleverd. In de bovenste twee rijen zal algemene informatie over het bed en de patiënt die er op ligt worden weergeven. Denk hierbij aan locatie, naam, leeftijd, datum van opname en behandelende dokter. De product owner wilde graag dat de gebruiker in het

dashboard alarmeringen kan instellen. Hiervoor zullen een viertal tegels rechts onderin het dashboard gereserveerd worden. Deze tegels beginnen leeg maar de gebruiker kan hier een alarm instellen door middel van de algoritme tool.

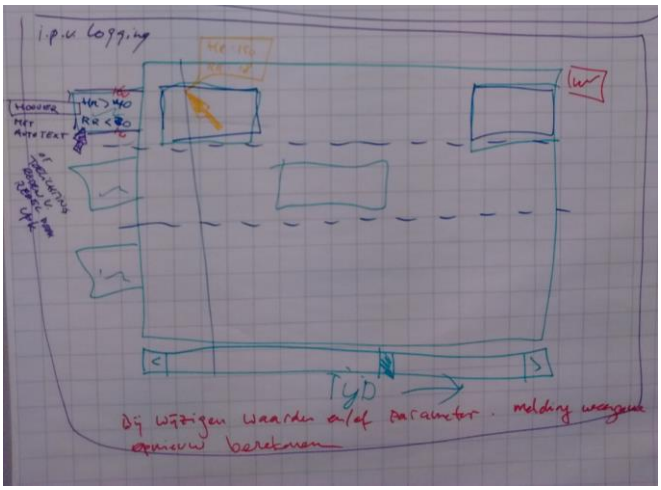


AFBEELDING 11: MOCKUP VOOR HET DASHBOARD OP PATIËNT NIVEAU.

Nog een wens is het inzichtelijk maken van historische alarmeringen en meldingen van het dashboard. De gebruiker moet kunnen zien wanneer een alarmeringen actief is geweest; ook wanneer er op dat moment geen alarm actief is. Hiervoor is de logging module bedacht welke in het midden onderin het dashboard geplaatst zal worden. Alle alarmeringen en meldingen worden hier in geplaatst met de tijd, oorzaak en prioriteit. Dit resulteert in een module met een heleboel regels tekst, iets wat tegen het principe van NewCompliance in gaat. Het dashboard moet alle informatie overzichtelijk weergeven met het liefst zo min mogelijk tekst. Om die reden wordt er op het dashboard ook veel gebruik gemaakt van icoontjes en verschillende kleuren om informatie te weergeven. De voorgestelde logging module zal daarentegen juist veel tekst bevatten. Toen werd het idee geopperd om een lijngrafiek te gebruiken voor de weergave. De lijnen zullen de parameters voorstellen die in het dashboard gebruik worden. Om een alarmering te weergeven zouden dan horizontale lijnen gebruikt kunnen worden die de grenswaarde voorstelt. Dit zal echter een zeer onoverzichtelijke grafiek worden door de hoeveelheid verschillende parameters en bijhorende schalen. Ook wordt het erg lastig om een alarmering duidelijk te weergeven omdat deze uit meerdere parameters kunnen bestaan met verschillende grenswaarden.

Ik kwam toen met het idee om de alarmeringen te weergeven in een tijdlijn. Elke alarmering krijgt in de tijdlijn een eigen baan. Op de momenten dat het alarm actief is geweest wordt een rood blok getekend. De alarmering wordt naast de tijdlijn op dezelfde hoogte van de bijhorende baan

ingesteld. Met deze oplossing kan de gebruiker snel en overzichtelijk zien wanneer een alarm actief is geweest. Deze oplossing is waarschijnlijk het makkelijkst te implementeren door de vier alarm modules en de tijdlijn module (voorheen logging module) samen te voegen tot één module, de 'alerts timeline-module'. In afbeelding 11 is de tekening van de tijdlijnmodule te zien die tijdens de brainstormsessie is gemaakt. Door aan de linker kant op de alarmering te klikken kunnen de 4 alarmeringen worden ingesteld. Er opent een pop-up waarin de naam, beschrijving en het algoritme van het alarm gewijzigd kan worden. Het algoritme wordt gemaakt met behulp van Blockly. Er zullen maar een beperkt aantal verschillende operaties mogelijk zijn binnen de programmeeromgeving en de uitkomst zal altijd waar of niet waar zijn. Het bouwen van de alert timeline zal de eerste stap zijn voor dit project. De algoritmes voor de alarmering zijn redelijk simpel. De kennis die in dit onderdeel wordt opgedaan kan vervolgens gebruikt worden om de algoritme tool verder toe te passen in de software.



AFBEELDING 12: FLIPOVER TIJDENS DE BESPREKING VAN DE LOGGING MODULE.

Naast de alarm en tijdlijn module was er ook de wens om een viertal 'custom modules' in het dashboard te plaatsen. Voor deze modules krijgt de gebruiker nog veel meer vrijheid om de module zo te bouwen of aan te passen naar zijn of haar wensen. Deze modules zullen echter niet vanuit het dashboard aan te passen zijn maar alleen door een gebruiker met extra rechten in de beheer pagina. Dit onderdeel is nog niet verder besproken, dit wordt bewaard voor later. Wel is de kans groot dat dit onderdeel erg zal gaan lijken op het ontwerp met triggers en effecten zoals in het begin van deze fase gemaakt is.

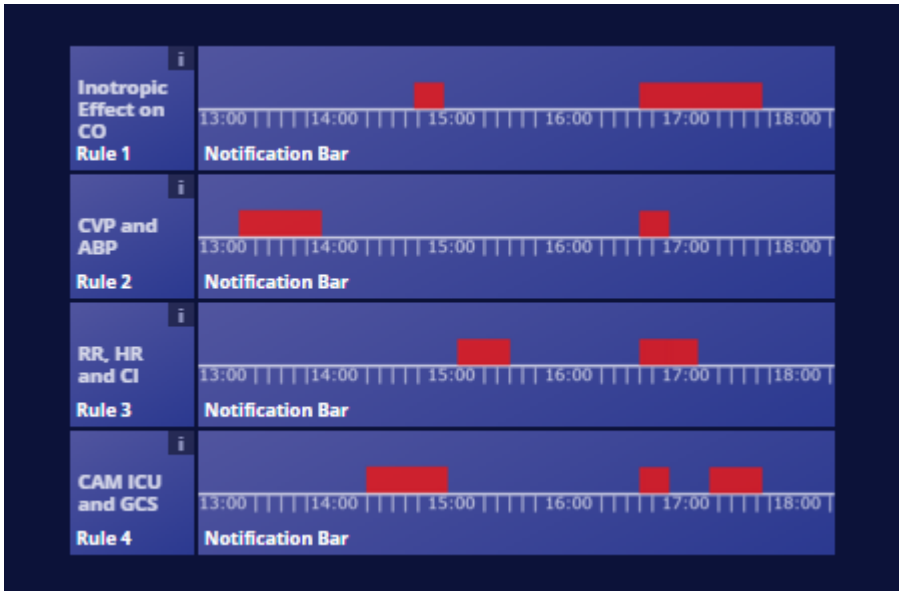
Met de opgedane informatie ben ik begonnen met het opstellen van de requirements. Hier is de volgende lijst uit voort gekomen:

ID	Requirement	MoSCoW
FR1	De gebruiker moet alarmeringen kunnen instellen in het dashboard op patiënt niveau.	M
FR2	Het moet mogelijk zijn een alarm te bouwen aan de hand van een threshold, trend of een combinatie van beide.	M
FR3	De gebruiker moet naast huidige alarmeringen ook de historische alarmeringen kunnen zien	M
FR4	De beheerder moet de logica van een nieuw type module grotendeels kunnen bepalen	M
NFR1	Niet-functioneel: De gebruiker moet binnen een dag getraind kunnen worden in het bouwen van een alarm.	S

Ik heb deze requirements uitgewerkt tot use-cases. Hier zijn slechts twee use cases uit voort gekomen. Zie hieronder een voorbeeld.

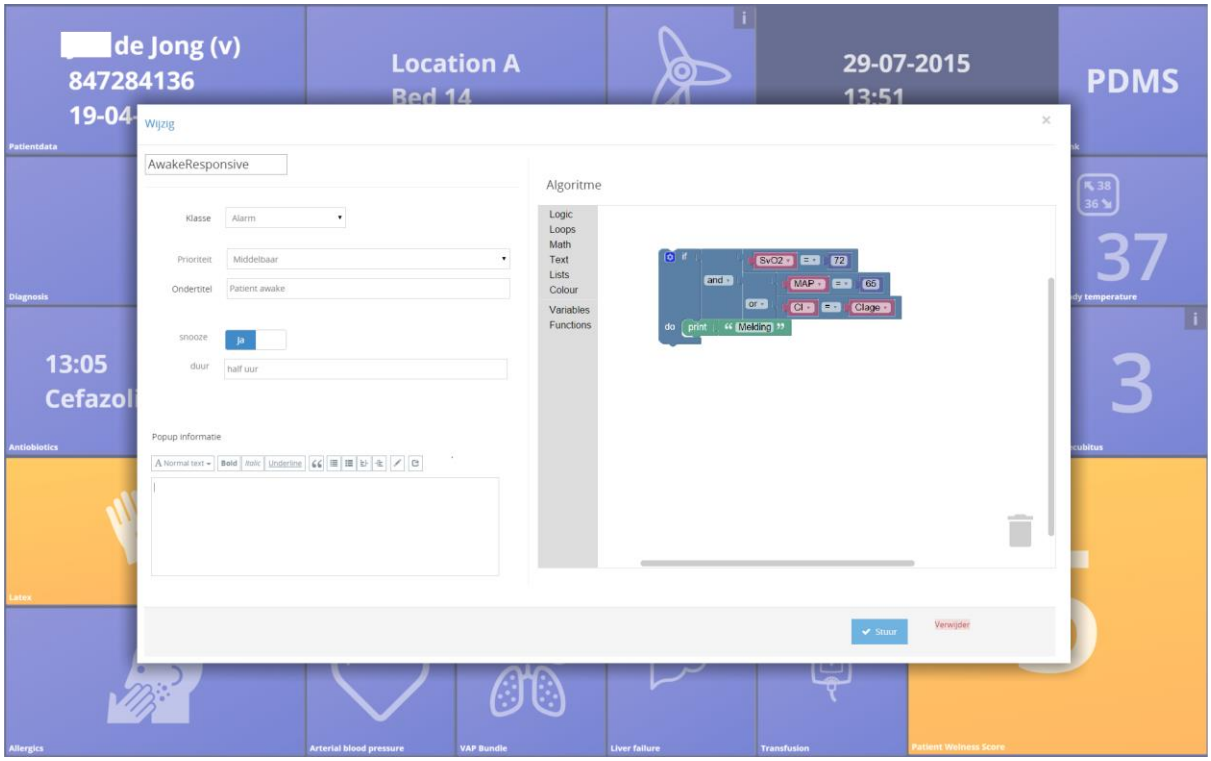
ID	UC2	FR1
Naam	Alert wijzigen	
Actoren	Gebruiker	
Precondities	Een dashboard dat de alerts timeline module implementeert is geopend.	
Verloop	1. De gebruiker klikt op een alarm 2. Een formulier voor naam, beschrijving en algoritme wordt geopend 3. De gebruiker vult de benodigde informatie in en klikt op de knop voor opslaan.	
Postcondities	De aanpassingen aan de alert zijn opgeslagen in de database.	
Alternatief verloop	Na stap 2 klikt de gebruiker op de annuleer knop. Het formulier wordt gesloten en de alert wordt niet gewijzigd/opgeslagen.	

Vervolgens ben ik aan de hand van deze use case beschrijvingen begonnen met het maken van een paar mockups. Voor de module is het belangrijk dat deze in dezelfde stijl als de rest het dashboard wordt gebouwd. Ik heb er toen voor gekozen om de illusie te wekken dat de tijdlijn module uit meerdere modules bestaat. Zo is het voor de gebruiker ook direct duidelijk dat de alarmeringen los van elkaar staan. Zie de mockup in afbeelding.



AFBEELDING 13: NIEUWE MOCKUP VAN DE ALERT TIMELINE MODULE.

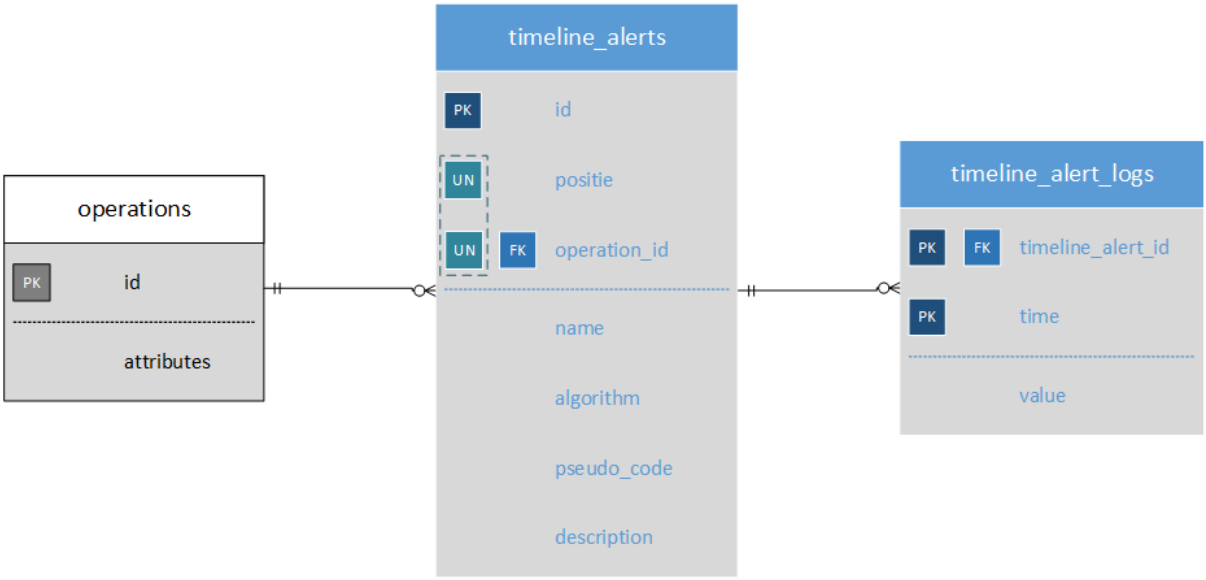
Als de gebruiker op een van de alert klikt wordt er een pop-up geopend waar de naam, beschrijving en andere informatie gewijzigd kan worden. Daarnaast bevat dit scherm een canvas waar door middel van Blockly een algoritme gebouwd kan worden.



AFBEELDING 14: MOCKUP VAN DE POP-UP WAARIN DE ALERT GEWIJZIGD KAN WORDEN.

Om de nieuwe module te maken zullen er een aantal aanpassingen in de database nodig zijn. Het moet namelijk mogelijk zijn om alarmeringen op te slaan. Hiervoor heb ik het ERD diagram gemaakt die te zien is in afbeelding .. voor het opslaan van de alerts heb ik de volgende attributen bedacht:

- id: unieke code om de alert te identificeren
- positie: de positie van de alert
- operation_id: vreemde sleutel naar de reeds bestaande operations tabel
- name: de naam van de alert die in de module weergeven wordt
- algorithm: het algoritme geëxporteerd vanuit Blockly
- pseudo_code: leesbare beschrijving van het algoritme om bij de module te weergeven
- description: beschrijvende tekst voor de alert



AFBEELDING 15: ERD VOOR HET OPSLAAN VAN DE ALERTS.

De timeline_alert_logs tabel wordt gebruikt om huidige en historische alarmeringen op te slaan zodat deze getekend kunnen worden in de tijdlijn. De vreemde sleutel timeline_alert_id linkt naar de timeline_alerts tabel en vormt samen met de time de primaire sleutel. De value is het resultaat voor het uitvoeren van het algoritme.

Ik heb voor de timeline_alerts tabel gekozen voor een gezamenlijke unieke sleutel met positie en operation_id. Zo forceer ik dat er niet meerdere alerts voor dezelfde plek in dezelfde kamer kunnen bestaan. In principe zou deze sleutel ook als primaire sleutel kunnen fungeren. Ik heb echter toch gekozen voor een uniek id, zodat maar één vreemde sleutel van timeline_alert_logs naar timeline_alerts nodig is.

4.3 Review

Deze fase bestond vooral uit het houden van meerdere brainstormsessies en besprekingen. NewCompliance was in deze periode nog erg druk met andere projecten en had daardoor niet veel aandacht aan het Thalea project besteed. Hierdoor heeft het veel tijd gekost voordat er een concreet plan was.

Uiteindelijk is uit de besprekingen de tijdlijn module ontstaan. Waarschijnlijk de meest complexe module ooit gemaakt voor het dashboard. Ik heb hiervoor een paar mockups gemaakt die een goede indicatie geven van het eindresultaat. De softwareanalist was erg enthousiast en had de mockups zelfs al laten zien aan de klanten. Ik kon hun nu natuurlijk niet teleurstellen, dus de druk was erop.

5 Construction fase 1

5.1 Plan

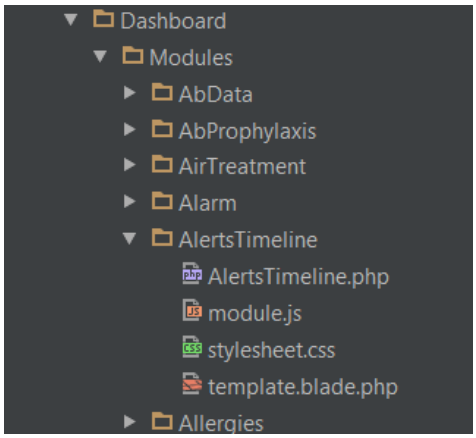
Het doel voor deze fase is het maken van een eerste versie van de tijdlijn module. De gebruiker moet een alert kunnen aanmaken en opslaan in de database. Met visueel programmeren kan de gebruiker een algoritme maken voor de alert. Vervolgens moet het algoritme worden uitgevoerd en het resultaat weergeven in een tijdlijn. Er hoeft in deze fase nog geen rekening gehouden worden met styling. Het belangrijkste is dat ik tijdens de demo aan het eind van de fase kan aantonen dat het maken van de module volgens de huidige aanpak mogelijk is.

Er worden twee API's gebruikt waar ik nog nooit eerder mee heb gewerkt: Blockly en vis.js. Het is daardoor lastig te voorspellen welke taken er allemaal uitgevoerd moeten worden voor een werkende tijdlijn module. De work items list bestaat daardoor slechts uit een aantal globale taken. Gaandeweg zal de work items list aangepast worden met specifiekere taken.

ID	Work item	MoSCoW
T1	Create new module: AlertsTimeline	M
T2	Create popup that implements Blockly	M
T3	Save/modify alerts in the database	M
T4	Build an interpreter that executes code exported by Blockly	M
T5	Implement vis.js to draw the timeline	M
T6	Style the module to look like the mockup	C

5.2 Uitvoering

De eerste stap is het maken van een nieuwe module. Een standaard module in de OK cockpit bestaat uit een php klasse, een javascript bestand, een CSS stylesheet en een blade template.



AFBEELDING 16: MAP EN BESTANDS INDELING VOOR DE DASHBOARD MODULES.

De tijdlijn module moet weten bij welke operation hij hoort. We overerven de ORModule klasse zodat we toegang hebben tot het operation attribuut. Ook zal onze module real-time informatie weergeven. Daarom implementeren we de updateable interface. Door deze interface te implementeren zal onze klasse meegenomen worden in de opbouw van het data object waarmee het dashboard van real-time data wordt voorzien. Ik moeten hiervoor wel de extractData methode aanmaken die de module data die ik aan het dataobject wil toevoegen teruggeeft. Elke module bevat ook een initialisatie methode. Hierin wordt de benodigde data uit de database opgehaald en eventueel berekend, zodat deze data met de extractData methode meegegeven kan worden. In mijn geval worden de alerts en alerts_logs opgehaald die bij de operation horen. Vervolgens wordt gecontroleerd of er voor elke positie (1 t/m 4) een alert bestaat. Is dit niet het geval dan wordt voor die positie een alert met standaard data aangemaakt. In de extractData worden de alerts samen met de alert logs meegegeven.

In de blade template kan de data die in het dataobject geplaatst is gebruikt worden om te weergeven. Vervolgens heb ik alle benodigde onderdelen gebouwd om een alert te maken, op te slaan en in te laden:

1. blade template voor de module.
2. AngularJS directive om een popup te openen.
3. algorithmBuilder. Javascript code dat zorgt dat Blockly wordt gestart. Het juiste algoritme wordt ingeladen en dat het algoritme juist wordt geëxporteerd.
4. algorithmController. De controller die alle requests afhandelt die te maken hebben met de algorithmBuilder waaronder het opslaan van de alert inclusief algoritme in de database.

Zo is de levenscyclus van een alert compleet waardoor het tijd wordt voor de grootste uitdaging van het project; het bouwen van een interpreter voor de geëxporteerde algoritmes.

Blockly kan de code in een aantal verschillende talen exporteren, waaronder: xml, json, javascript en php. Aangezien de OK cockpit is gebouwd met php is dit de meest logische export om te gebruiken. Maar hoe zorg ik ervoor dat deze code wordt uitgevoerd? Een mogelijkheid is het gebruiken van de eval functie. Deze functie voert een string met php code uit. Het gebruik van de eval is echter erg

gevaarlijk omdat het de uitvoer van ongeautoriseerde code mogelijk maakt. Zeker in combinatie met user input is het gebruik van eval ongewenst.

Een voorbeeld: Stel we hebben op een website een pagina waar de gebruiker wiskundige opdrachten kan uitvoeren. De gebruiker moet eerst ingelogd worden. Als de gebruiker is ingelogd wordt de globale variable $_SESSION['id'] op de id gezet die hoort bij de gebruiker. Deze variabele wordt vervolgens gebruikt op elke pagina om te controleren of de gebruiker is ingelogd en of de gebruiker de juiste rechten bezit. Als de gebruiker is ingelogd kan hij een rekensom invoeren in het invoerveld. Deze invoer wordt uitgevoerd door de eval functie en het resultaat wordt weergeven op de website.

Niets aan de hand met dit soort input:

index.php?opgave=3x5
index.php?opgave=6*9

Maar stel dat iemand de volgende input geeft.

index.php?opgave=3x5;$_SESSION['id']=200

Dit betekend dat de gebruiker ineens is ingelogd op een ander account. We kunnen hier dan ook spreken over een serieus lek in de software.

Om dit soort ongewenste praktijken tegen te gaan moet de invoer voldoende worden geverifieerd. Het zal echter erg lastig zijn om een input dat een algoritme betreft te verifiëren op ongewenste code. Liever vermijden we het gebruik van eval() in zijn geheel. Ik ben toen gaan kijken naar de json export. Deze export wordt op dit moment al gebruikt om het algoritme op te slaan zodat deze in Blockly kan worden ingeladen. In het json formaat wordt het algoritme hiërarchisch opgeslagen. De geëxporteerde json kan omgezet worden in genestelde php objecten. Elk object stelt een blokje voor van Blockly. Het algoritme kan vervolgens uitgevoerd worden door voor elk blok de bijhorende logica uit te voeren. Om deze oplossing te implementeren moet voor elk type blok een methode geschreven worden. Daarnaast moet een stuk code zorgen dat per blok type de juiste methode wordt uitgevoerd. Dit kan door een grote reeks if-statements of switch-cases aan te maken. Maar een mooiere oplossing is om gebruik te maken van de flexibiliteit van php. Php bied namelijk de mogelijkheid om variabele functies aan te roepen. Zie onderstaande code:

```
function foo() {

    echo "In foo()<br />\n";

}


$func = 'foo';

$func();        // This calls foo()
```

Aan de hand van een string wordt de juiste methode aangeroepen. In dit voorbeeld wordt de methode foo aangeroepen. Door de methode namen gelijk te stellen aan het type attribuut per block in het geëxporteerde algoritme, kan zo gemakkelijk gezorgd worden dat de juiste methode wordt aangeroepen.

De volgende vraag is waar deze interpreter toegepast gaat worden. Mijn plan was om de interpreter in de alert timeline module te plaatsen. Doordat de module updateable is zal daardoor de interpreter met elke update van het dataobject van het dashboard uitgevoerd worden. Maar na een performancetest van de interpreter bleek dat bij het bestaan van veel alerts dit veel tijd kost. Dit veroorzaak vertraging in het updaten van het dashboard. Omdat het van groot belang is dat de data op het dashboard real-time is, het gaat immer om mensenlevens, vormt een vertraging van een paar seconden al een groot probleem. Sylvester heeft mij toen aangeraden om de interpreter in een service te laten draaien. Deze service zal via de command line worden gestart en zal continue voor elk ingestelde alert berekenen wat de uitkomst van het algoritme is. Bij een verandering ten opzichte van de vorige uitkomst wordt er een nieuwe log aangemaakt. Deze service heb ik de timeline alert reader genoemd.

Voor de tijdlijn wordt gebruik gemaakt van de javascript API vis.js. Deze API wordt reeds in de OK cockpit gebruikt om een tijdlijn te tekenen. De belangrijkste redenen dat vis.js wordt gebruik is omdat het snel is en makkelijk aan je eigen stijl aan te passen. Dit komt doordat de gehele tijdlijn wordt opgebouwd uit dom elementen die slim zijn geclassificeerd. Met CSS kan vervolgens de stijl, vrijwel zonder restricties, worden aangepast.

Vis.js tekent de tijdlijn aan te hand van een aantal opties en een dataset. De dataset bestaat uit een array met items. Elk item bevat een id, starttijd, eindtijd en groep. Items van dezelfde groep worden in dezelfde baan geplaatst. De structuur van de dataset van vis verschilt sterk met de data die binnenkomt. In de tabel hieronder wordt dezelfde data opgeslagen maar in een ander structuur.

Change of value	**Vis.js dataset**

- Groep1
  o Item1
  ▪ Tijd: 12:00
  ▪ Waarde: true
  o Item2
  ▪ Tijd: 12:30
  ▪ Waarde: false
  o Item3
  ▪ Tijd: 13:00
  ▪ Waarde: true
  o Item4
  ▪ Tijd: 13:15
  ▪ Waarde: false
- Groep2
  o Item1
  ▪ Tijd: 12:30
  ▪ Waarde: true
  o Item2
  ▪ Tijd: 14:30
  ▪ Waarde: false
	- Item1
	  o Starttijd: 12:00
	  o Eindtijd: 12:30
	  o Groep: 1
	- Item2
	  o Starttijd: 13:00
	  o Eindtijd: 13:15
	  o Groep: 1
	- Item3
	  o Starttijd: 12:30
	  o Eindtijd: 14:30
	  o Groep: 2

Het was niet gemakkelijk om het algoritme te schrijven voor het juist omzetten van de data. Voor de geïnteresseerde is hieronder het algoritme te zien.

```
for (var i = 0; i < alerts.length; i++) {
  if (alerts[i].alertLog) {
    var alert = alerts[i];
    var item = {};
    var firstValue = true;
    var isItemFinished = true;
    for (var j = alert.alertLog.length - 1; j >= 0; j--) {
      var alertLog = alert.alertLog[j];
      if (firstValue) {
        if (alertLog.value == true) {
          item.end = currentTime;
        } else {
          item.end = alertLog.time;
          isItemFinished = false;
        }
        firstValue = false;
      }
      if (!isItemFinished) {
        if (alertLog.value == 1) {
          item.start = alertLog.time;
          item.id = i + "" + j;
          item.group = "alert" + (i + 1);
          items.push(item);
          item = {};
          isItemFinished = true;
        }
      } else {
        if (alertLog.value == false) {
          item.end = alertLog.time;
          isItemFinished = false;
        } else {
          item = {};
        }
      }
    }
  }
}
```

AFBEELDING 17: HET ALGORITME DAT DE BINNENGEKOMEN DATA OMZET NAAR DE STRUCTUUR VAN VIS.JS

Na het omzetten van de data met het algoritme werd de data correct in de tijdlijn weergeven. Toen werd het tijd om de tijdlijn te stylen met CSS. Dit stylen heeft veel tijd gekost. Alles moet netjes meeschalen wanneer de schermgrootte wordt aangepast. Voor een aantal html elementen heb ik het juist meeschalen moeten forceren door gebruik te maken van event listeners. Event listeners starten een functie zodra er een event gebeurt. In ons geval zodra de venstergrootte veranderd. In de functie wordt vervolgens gezorgd dat het element de juiste grootte krijgt.

Na deze make-over met CSS begint de module echt tot leven te komen. Zie hieronder het het resultaat na het stylen met CSS.

5.3 Test

Ik heb de applicatie op twee manieren getest. Eerst heb ik unittests gemaakt voor de methodes in de TimelineAlerts module. Deze tests zijn gemaakt met PHPunit. PHPunit wordt ook door Laravel ondersteund is daardoor makkelijk in gebruik. Zie hieronder een voorbeeld van een unittest. Deze unittest controleert of de methode de juiste alerts uit de database ophaald.

```php
public function testRetrieveAlerts() {
    $alertTimeline = new NewCompliance\Dashboard\Modules\AlertsTimeline\AlertsTimeline($this->internal, $this->room, $this->operation);
    $alertTimeline->init();

    $alertTimeline->retrieveAlerts();

    $this->assertEquals($this->alerts, $alertTimeline->getFieldValue('alerts'));
}
```

Vervolgens kan met een command alle tests die zijn gedefinieerd worden uitgevoerd. Na het uitvoeren van de test kreeg ik als resultaat dat 54 van 56 tests succesvol zijn. De twee onsuccesvolle test bleken te zijn ontstaan doordat deze unit tests niet meer up to date zijn. Het falen van de twee tests had niks met mijn code te maken.

Naast de unit tests heb ik ook use case tests uitgevoerd. De drie use cases heb ik uitgewerkt naar testcases zoals in onderstaand voorbeeld:

LOGISCHE CASE Wijzig alarm	**UC2**
actie	**Gewenste resultaat**
Gebruiker opent een dashboard dat de alert timeline module implementeert	Het systeem toont de tijdlijn module
De gebruiker klikt op een alarm	Het systeem opent een formulier
De gebruiker vult het formulier in	-
De gebruiker klikt op opslaan	Het systeem slaat het alarm op in de database

5.4 Review

Ik ben zeer positief over het verloop van deze fase. Het doel voor deze fase is behaald; een functionerende tijdlijn module. Ik ben uiteindelijk zelfs verder gekomen dan ik had verwacht. Zo is het tijdens deze fase gelukt om de module te stylen. Iets waarvan ik had verwacht dat dit in de 2e constructie fase pas aan bod zou komen. Hierdoor zag het prototype er meteen een stuk meer ‘af’ uit, wat uiteraard erg fijn is.

Over het product ben ik erg te spreken. Ik heb met het prototype aangetoond dat de technische aanpak werkt. Een minpuntje is dat de huidige interpreter niet erg efficiënt te werk gaat. Het kost daardoor relatief veel tijd om een simpel algoritme uit te voeren. Ik kan op dit moment helaas nog niet iets bedenken om hier iets aan te verbeteren.

De module is echter verre van af zo bleek uit de review sessie waarin ik een demo heb gegeven van dit eerste prototype. De reacties waren erg positief waardoor iedereen heel enthousiast met nieuwe ideeën kwam. Uit deze nieuwe ideeën en verbeteringen hebben we direct een selectie gemaakt. Deze selectie zal in de volgende fase worden uitgevoerd.

6 Construction fase 2

6.1 Plan

Deze fase is het doel om het mogelijk te maken parameters uit de database te gebruiken als variabelen in het algoritme. Van deze parameters wordt ook de historische data opgeslagen. Deze historische data wil ik gebruiken om historische alerts te berekenen. Op het moment dat een alert wordt aangepast moet de gebruiker de alerts uit het verleden in de tijdlijn kunnen zien.

Op dit moment zijn bijna alle modules van het type ORModule. Deze fase wil ik een nieuw type module introduceren; de ICBEDModule. De ICBEDModule bevat in plaats van een operation een patiënt_stay. patiënt_stay wordt een nieuwe tabel in de database die voor de IC cockpit de plaats inneemt van operations. De tabel geeft informatie van een patiënt verblijf en kan gezien worden als een koppeltabel voor patiënten en intensive care bedden.

De rest taken die zijn ingepland voor deze fase komen bijna allemaal rechtstreeks uit de review sessie van de vorige fase.

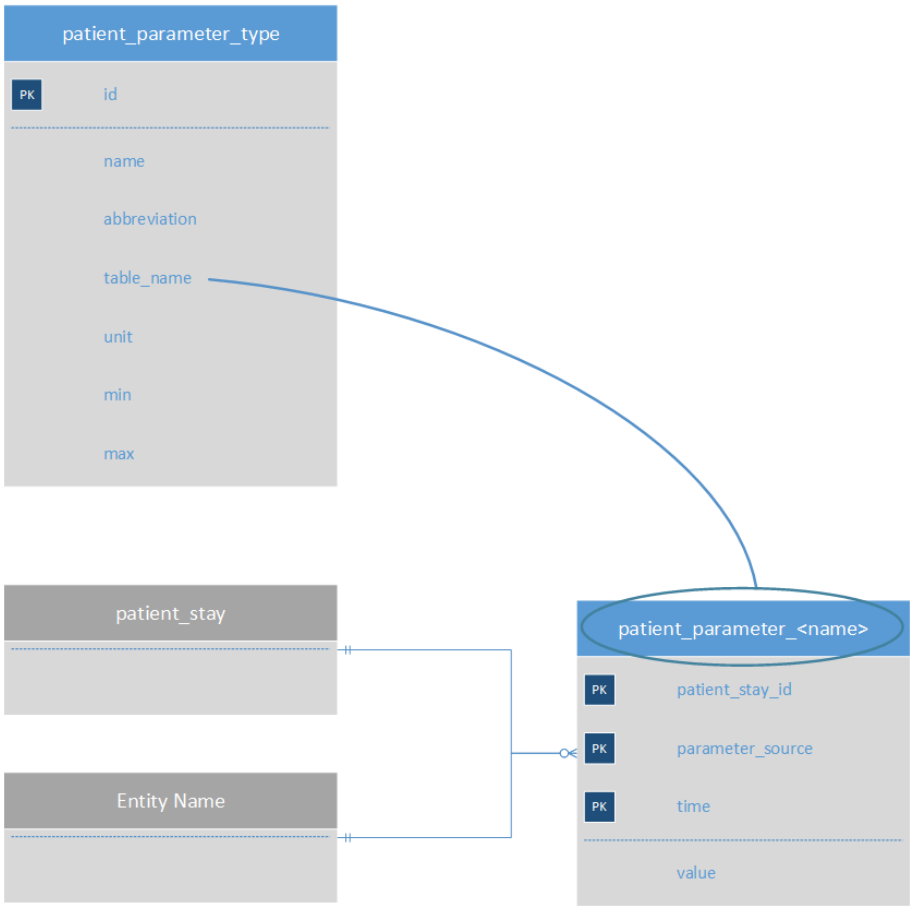
work item list:

ID	Work item	Priority (MoSCoW)
M1	Create new database design for patient_stays instead of operations	M
M2	Create new module type ICBED that uses patient_stays instead of operations	M
T21	Create a new 'patient parameters' block in Blockly with a dropdown with available patient parameters	M
T22	Enhance the algorithm interpreter to read parameters from the database	M
T23	Make the module keyboard independant	C
T24	Create method that exports the Blockly algorithm as pseudo code.	S
T25	Use pseudo-code as sub-title for an alert	S
T26	Compute alerts based on historical data	M
T27	The timeline should always show the current time as end time	M
T28	Give the user feedback on saving an alert	S
T29	Turn alert and corresponding bars in the timeline red when currently true	M
T30	Custom Blockly blocks that enable users to calculate trends	M
T31	Turn alert timeline bars yellow when the alert is finished. When the user interacts with the module return to normal color.	S

6.2 Ontwerp

Deze fase maak ik de eerste stap naar het omzetten van de OK cockpit naar IC cockpit. In de OK cockpit wordt de inhoud van het dashboard bijna geheel bepaald door de bijhorende operatie. In de nieuwe IC cockpit moet dit veranderen naar een patiënt verblijf. Een patiënt verblijf is een patiënt op een intensive care bed gedurende een tijdsperiode.

Daarnaast willen we parameters die bij een patiënt verblijf horen opslaan. Zie de ERD hieronder:



AFBEELDING 18: ERD VAN PATIËNT PARAMETERS.

Voor elke parameter wordt een nieuwe tabel aangemaakt. Hierin wordt elke ontvangen waarde van de parameter opgeslagen. Deze tabel heeft een vreemde sleutel naar patiënt verblijf en een vreemde sleutel naar parameter_source. De parameter_source tabel bevat de verschillende machines die parameters kunne, produceren. Deze tabel zal pas in de live versie gebruikt worden maar is voor de compleetheid hier alvast meegenomen.

Om makkelijk door alle parameters heen te kunnen zoeken en om te weten welke parameters er bestaan wordt er nog een extra tabel aangemaakt. Hierin wordt de naam van een parameter opgeslagen met daarbij de naam van de bijhorende tabel. Daarnaast wordt wat informatie over de parameter opgeslagen zoals de eenheid en de minimale en maximale waarde.

Voor deze fase is ook een twee nieuwe requirements gedefinieerd:

ID	Requirement
FR5	Kleur een alarm rood wanner deze alarmerend is.
FR6	Kleur een alarm geel wanneer er een alarmering is geweest totdat de gebruiker aangeeft het alarm gezien te hebben.

Vanuit de FR6 requirements is ook een nieuwe use case ontstaan:

Use case ID	Use Case	Beschrijving	Requirement ID
UC3	Acknowledge alarmering	De gebruiker geeft aan dat de alarmering gezien is.	FR6

En de hier bijhorende use case beschrijving:

ID	UC3
Naam	Acknowledge alarmering
Beschrijving	De gebruiker geeft aan dat de alarmering gezien is.
Actoren	Gebruiker
Precondities	• Een dashboard dat de alerts timeline module implementeert is geopend. • In de database staat een actieve alarmering
Verloop	1. Het systeem toont de tijdlijn module 2. Het systeem detecteert dat een alarmering actief is 3. Het systeem kleurt de tijdlijn geel 4. Het systeem weergeeft de acknowledge knop 5. De actor klikt op de acknowledge knop 6. Het systeem kleurt de tijdlijn terug naar de oorspronkelijke kleur
Postcondities	-
Alternatief verloop	-

6.3 Uitvoering

De eerste stap voor deze fase is het aanpassen van de software zodat dashboards gebouwd kunnen worden die een patiënt verblijven gebruiken in plaats van operaties. Om dit te bewerkstelligen was het onder andere nodig om een nieuw type module aan te maken. De ICBEDModule erft net als de ORModule de RoomModule over en is daardoor vergelijkbaar met de ORModule. In de ModuleCollection klasse wordt gezorgd dat in elke module die ICBEDModule overeft het patiënt verblijf wordt opgeslagen. Ik heb de klasse van de tijdlijn module aangepast dat deze nu de ICBEDModule overerft. De tijdlijn module is de eerste module van het type ICBED. Door deze aanpassing hebben we in de tijdlijnmodule nu beschikking over het patiënt verblijf. Met behulp van het patiënt verblijf kunnen we de juiste patiënt parameters ophalen. En zo kom ik aan bij de volgende stap. Ik ga nu zorgen dat de software algoritmes met patiënt parameters ondersteund.

Ik begin bij de algoritme bouwer. Ik moet Blockly uitbreiden met een nieuw type blok. Dit blok lijkt op het standaard variabelen blok maar moet in de dropdown alle mogelijke patiënt parameters weergeven. Gelukkig hebben de makers van Blockly er rekening mee gehouden dat gebruikers zelf nieuwe blokjes willen maken. Hierdoor kostte het niet veel tijd om het nieuwe variabelen blokje aan te maken. De dropdown moet gevuld worden met alle mogelijke parameters. Hiervoor heb ik een ajax request gebouwd die via een nieuwe methode in de algorithmController alle parameters uit de parameter types tabel haalt.

Vervolgens moet de interpreter worden aangepast zodat bij blokken van het type patiënt_parameter de juiste code wordt uitgevoerd. Om vervolgens de juiste waarde uit de database te halen zou ik normaalgesproken de model klasse gebruiken van de tabel. Echter staan de patiënt parameters in verschillende tabellen. Ik heb daarom een extra klasse voor de parameters models gemaakt. Deze klasse werkt als een normale model maar past zich aan, aan de hand van de parameter naam. Dit maakt het werken met patiënt parameters een stuk gemakkelijker. Deze nieuwe klasse gebruik ik in de interpreter om de laatste waarde van de parameter met het juiste patiënt verblijf en type op te halen uit de database.

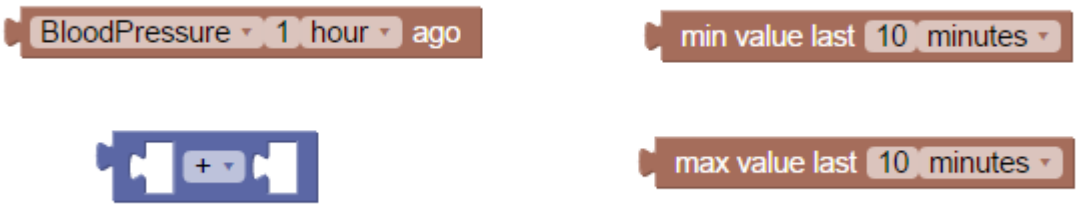
De volgende taak is het terugrekenen van historische alerts. Hiervoor moet de timeline alert reader worden aangepast. Allereerst heb ik gezorgd dat zodra een algoritme verandert, alle alert logs die daarbij horen worden verwijderd. Vervolgens gebruikt de reader dit gegeven als indicatie dat een algoritme is verandert. De reader controleert of er minstens een alert log bestaat. Zo niet dan zullen de historische alerts berekend worden. Bij het opstarten van de service kan optioneel een precisie en periode opgegeven worden. De precisie bepaald per hoeveel minuten de uitkomst wordt berekend. De periode bepaald hoeveel uur wordt teruggerekend. Het terugrekenen wordt gedaan door een for-loop waarin constant x (precisie) aantal minuten van een datum-tijd variabele wordt afgetrokken totdat de terugkijk periode is bereikt. Binnen de loop wordt de interpreter aangeroepen. Aan de interpreter wordt de datum-tijd variabele van de loop meegegeven. Binnen de interpreter wordt deze variabele gebruikt om de juiste waarden voor de patiënt variabelen op te halen.

Dit waren de belangrijkste taken voor deze fase. De rest van de taken zijn relatief wat kleiner. Toch staan er nog een paar taken die de prioriteit ‘must-have’ hebben meegekregen. Bijvoorbeeld dat de alert en de bijhorende tijdlijn rood moet kleuren wanneer de alert ‘true’ is. Ik heb hierbij direct ook de taak ‘Turn alert timeline bars yellow when the alert is finished. When the user interacts with the module return to normal color’ opgepakt omdat deze er sterk mee te maken heeft. In plaats van het verkleuren van geel naar de normale kleur wanneer er een interatie plaatsvind is er na overleg voor gekozen om hier een knop van de maken. De ‘acknowledge’ knop komt tevoorschijn wanneer een

alarm is afgelopen. Op dat moment kleurt ook de tijdlijn geel. Als de gebruiker op de knop drukt krijgt de tijdlijn zijn normale kleur weer terug.

De laatste ‘must have’ taak voor deze fase is het toevoegen van een aantal blokken voor trendberekeningen. Een trend kan uiteraard op veel verschillende manier berekend worden. Wat voor de alarmeringen vooral interessant is de stijging over een periode. Dit kan heel simpel worden berekend door het verschil tussen de waarde van parameter x tijd geleden en de huidige waarde te nemen. Hiervoor zijn twee nieuwe blokken in de algoritme bouwer nodig.

Een andere soort trend die interessant is is hoe stabiel een parameter is. Wanneer de hartslag van een patiënt zeer afwisselend is kan dit een belangrijke indicator zijn voor de gezondheid van de patiënt. Hiervoor zouden we het verschil tussen de minimale en maximale gedurende een tijdperiode kunnen nemen. Hiervoor hebben we nog 2 nieuwe blokken nodig: de min en de max in een periode.



AFBEELDING 19: NIEUWE BLOCKLY BLOKKEN VOOR TRENDBEREKENINGEN

Ik verwacht dat er nog meer interessante berekening mogelijk moeten zijn maar daar kan ik pas aan beginnen als de wensen daarvoor beter zijn gespecificeerd. De softwareanalist heeft besloten dat mogelijkheden voor het berekenden van trends voorlopig voldoende is.

Alle ‘must have’ taken zijn voltooid en dus begin ik nu aan de ‘should have’ en ‘could have’ taken.

Ik heb ervoor gezorgd dat de tijdlijn altijd de huidige tijd als eindtijd heeft, zelfs wanneer er wordt ingezoomd. Dit gaf echter wel problemen met het inzoomen in de tijdlijn. Normaal gesproken wordt er door vis.js in de tijdlijn ingezoomd naar de locatie van de pointer. Doordat op het moment dat wordt ingezoomd de eindtijd opnieuw naar de huidige tijd wordt gezet ontstond er een ongewenste situatie. Wanneer de pointer dicht bij de eindtijd is wordt er met de normale snelheid ingezoomd. Zodra de pointer meer richting het begin van de tijdlijn wordt verplaats wordt de zoom snelheid steeds langzamer. Ik heb dit opgelost door vis.js uit te breiden met een nieuwe config optie. Bij het initialiseren kan als optie ‘zoomCenter’ worden meegegeven. Als deze optie op true wordt gezet zal er altijd naar het midden van de tijdlijn worden gezoomd, met als resultaat dat waar de pointer zich ook bevind de zoom snelheid hetzelfde zal zijn.

Uit de reviewsessie van de vorige fase bleek dat het gewenst is dat de gebruiker kan zien om wat voor alert het gaat zonder deze te openen. Hieruit is het idee ontstaan om als sub titel de pseudocode van het algoritme te weergeven. Blockly bied helaas geen mogelijkheid om algoritmes als pseudo code te exporteren dus dit zal ik zelf moeten bouwen. Ik heb tijdens het bouwen veel afgekeken bij de reeds bestaande export methodes. Uiteindelijk wordt er een string opgebouwd met de leesbare pseudo code. De export methode is onvolledig. Lang niet alle blokken van Blockly worden ondersteund. Dit betekend dat wanneer ik nieuwe blokken toevoeg aan de algoritme bouwer, ik ook de pseudo code export methode moet aanpassen.

6.4 Test

De fase zijn net als bij de vorige fase unit tests gebouwd. Tijdens de uitvoer bleek alle test succesvol voltooid te zijn.

In deze fase is er een nieuwe use case bijgekomen en dat betekend uiteraard een nieuwe testcase:

LOGISCHE CASE acknowledge alarmering	**UC3**
actie	**Gewenste resultaat**
Gebruiker opent een dashboard dat de alert timeline module implementeert	Het systeem toont de tijdlijn module
Een alarm wordt actief en stopt daarna	Het systeem kleurt de tijdlijn geel en weergeeft de acknowledge knop
De gebruiker klikt op de acknowledge knop	Het systeem kleurt de tijdlijn terug naar de oorspronkelijke kleur en de acknowledge knop verdwijnt

6.5 Review

Ik ben wederom erg tevreden over het verloop. De fase had als doel het voltooien van de tijdlijn module. Dit doel heb ik bereikt, alle gewenste functionaliteiten zijn aanwezig. Ik heb wel mijn twijfels bij het onderdeel trendberekening, dit voelt nog onvolledig. Het is echter erg lastig te achterhalen wat voor berekeningen de gebruiker wil maken. Hiervoor zou ik concrete voorbeelden willen zien. Deze voorbeelden heb ik helaas niet kunnen vinden en ook onze softwareanalist heeft mij deze voorbeelden niet kunnen aanleveren.

7 Construction fase 3

7.1 Plan

Dit is de laatste constructiefase van het project. In de afgelopen 2 fases heb ik gewerkt aan de tijdlijn module. De module is nu af en dus is het tijd om aan het 2e deel van het project te beginnen. Al sinds de elaboration fase bestaat het plan voor de zogenoemde custom modules. Modules die bijna geheel aanpasbaar zijn door middel van visueel programmeren. De plannen waren echter nog lang niet concreet.

Deze fase zal dan ook voor een groot deel bestaan uit het uitwerken van ideeën voor deze nieuwe module. De opzet van het verslag zal daardoor voor deze fase iets veranderen. In de huidige paragraaf planning wordt normaal gesproken een plan gemaakt door kiezen van taken uit de work items list. Deze lijst is echter pas opgesteld na het maken van de ontwerpen en zal daarom ook pas aan het eind van de design paragraaf getoond en besproken worden.

7.2 Ontwerp

Er moest een plan gemaakt worden voor de verdere invulling van mijn afstuderen. Er was nog maar 1 constructie fase over, wat betekend dat er nog 3 weken zijn om in te vullen. Ik heb een brainstormsessie ingepland waar we hierover gingen nadenken.

Al gauw ging de bespreking over de eerder genoemde custom modules. Voorheen was het plan om de gebruiker bij deze modules bijna ongelimiteerde vrijheid te geven. Dit zou erg lastig zijn om te bewerkstelligen en daarom niet realistisch voor de komende 3 weken. Toen zijn we gaan nadenken welke bestaande modules meer aanpassingsmogelijkheden kunnen gebruiken. Toen bleek dat de modules voor de vitale parameters (hartslag, bloeddruk etc.) hier mogelijkheden in missen. Het gebeurt al vaak dat klanten net verschillende instellingen en opties willen, die op dit moment nog niet mogelijk zijn.

zo is het idee ontstaan voor de abstracte numerieke module. Deze module bevat uit zichzelf geen logica en geen lay-out. Maar kan door de gebruiker gevormd worden naar zijn. haar wens. De nieuwe requirements vind je hieronder en specificeren de reeds bestaande FR4 requirement:

ID	Requirement
FR4	De beheerder moet de logica van een nieuw type module grotendeels kunnen bepalen (verder specificeren)
FR4a	De beheerder kan bepalen onder welke condities de achtergrondkleur van de module verandert
FR4b	De beheerder kan bepalen onder welke condities de achtergrond icoon van de module verandert
FR4c	De beheerder kan bepalen onder welke condities de richting van de pijltjes van de module verandert
FR4d	De beheerder kan bepalen onder welke condities de kleur van de pijltjes van de module verandert
FR4e	De beheerder kan bepalen welke waarde in de module wordt weergegeven
FR4f	De beheerder kan bepalen onder welke condities de module alarmerend is (wordt gebruikt door vitale parameters module)

Bij FR4 horen twee nieuwe use cases:

Use case ID	**Use Case**	**Beschrijving**	**Requirement ID**
UC4	Bekijk abstracte numerieke module	De gebruiker ziet informatie over de ingestelde module	FR4
UC5	Wijzig abstracte numerieke algoritme	De gebruiker wijzigt een abstracte numerieke module	FR4

Vervolgens ben ik de use cases gaan uitwerken tot use case beschrijvingen. Daarna kon ik verder met het uitwerken van de implementatie.

Met behulp van een uitgebreide versie van de algoritmebouwer moet de gebruiker de module opbouwen. Hiervoor zal een hele reeks nieuwe blokken voor Blockly gemaakt worden die gebruikt moeten worden om de module visueel te manipuleren. Wellicht volgen er in de toekomst nog meer maar op dit moment zijn de volgende manipulaties bedacht:

- Value: De waarde die in het midden van de module getoond moet worden. Hier wordt normaal gesproken een patiënt variabele aan gekoppeld.
- BackgroundColor: De achtergrond kleur van de module.
- BackgroundIcon: Een icoon dat in de achtergrond wordt geplaatst.
- Min/max: Grenswaarden die in de module worden weergeven.
- Triangle: Pijltjes waarmee kan worden aangegeven of de waarde stijgt of daalt.

Dan is er nog de vraag waar deze modules gebouwd kunnen worden. Omdat het bouwen van de algoritmes voor deze modules nog een stap complexer is dan die van de alerts timeline is ervoor gekozen om het aanpassen niet mogelijk te maken voor de gewone gebruiker, maar alleen voor de

zogenaamde superuser. De superuser heeft in tegenstelling tot de gewone gebruiker naast de dashboard ook toegang tot de beheertool en analyse tool. De algoritmebouwer voor de abstract numerical module zullen we dan ook niet in het dashboard plaatsen maar in de beheertool. Nog specifieker; op de module beheer pagina. In het formulier waar de gebruiker de module kan aanpassen zal in het geval van een abstract numerieke module een extra veld verschijnen. In dit veld kan de json export van een algoritme geplakt worden. Zo kan gemakkelijk een bestaand algoritme, dat bijvoorbeeld via de mail verstuurd is, geïmporteerd worden. Indien de gebruiker zelf een algoritme wil maken of het huidige algoritme wil aanpassen kan hij/zij op knop naast het veld klikken. Er opent dan een pop-up met daarin de algoritmebouwer. Grotendeels hetzelfde als de algoritmebouwer van de alertstimeline module alleen heeft de gebruiker toegang tot andere blokken. Zie de mockup in afbeelding

AFBEELDING 20: ONTWERP VAN TWEE ABSTRACTE NUMERIEKE MODULES.

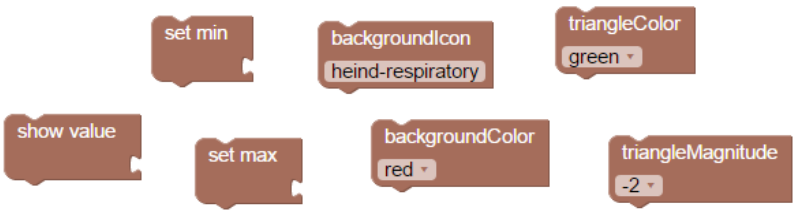
Vervolgens kon ik de work items list voor de rest van deze fase opstellen. Het resultaat zie je hieronder:

ID	Work item	MoSCoW
N1	Create new module: AbstractNumerical	M
N2	Change module config to show a new field when editing an AbstractNumerical module	M
N3	Create a popup that implements the algoritmBuilder	M
N4	Create new Blockly blocks and extend the interpreter to support new blocks	M
N5	Create template and Angular directives to manipulate the lay-out	M

7.3 Uitvoering

Ik ben begonnen met het maken van de nieuwe module op dezelfde manier als de alertstimeline module. Deze nieuwe module heb ik de AbstractNumerical module genoemd. Vervolgens heb ik de pagina voor het wijzigen van modules aangepast zodat deze een extra tekstveld weergeeft wanneer er een AbstractNumerical module wordt gewijzigd. Daarna heb ik de pop-up gemaakt die de algoritmebouwer implementeert.

Toen was het tijd om de nieuwe Blockly blokken toe te voegen. Ik heb de nieuwe blokken eerst in Blockly gebouwd en heb daarna de bijhorende logica in de interpreter gemaakt. De interpreter moest wel wat aangepast worden. Voorheen hadden we altijd maar te maken met een enkel resultaat. Deze werd aan het einde van het uitvoeren van het algoritme als return value gegeven. Nu kan het algoritme meerdere resultaten hebben. In veel gevallen zullen er meerdere manipulaties voor de module uit het algoritme volgen. Ik heb ervoor gezorgd dat alle resultaten in een associatieve array word opgeslagen. Deze array wordt vervolgens na het uitvoeren van het algoritme teruggegeven.



AFBEELDING 21: NIEUWE BLOCKLY BLOKKEN VOOR MODULE MANIPULATIES

In de klasse van de AbstractNumerical module roep ik de vernieuwde interpreter aan. De verkregen array wordt vervolgens aan de template meegegeven. In de front end heb ik een reeks van Angular directives gemaakt die ervoor zorgen dat de manipulaties die in de array staan opgeslagen worden doorgevoerd. Alles werkt en zo was de nieuwe module al voltooid. Dit ging vele malen sneller ik had verwacht en daar was er nog tijd over voor een uitbreiding op de AbstractNumerical module.

Van de softwareanalist heb ik de mockup voor het dashboard op centraal niveau ontvangen. Hierin werd de AbstractNumerical module die ik zojuist gebouwd had al genoemd. Dit was echter wel een bijzondere variant ervan. In de mockup stelt een rij van modules een patiënt voor. Bij elke patiënt staan twee modules, dat AbstractNumerical modules lijken, gedefinieerd. Maar wat deze module bijzonder maakt is dat hij de vorm aanneemt van een van de achterliggende AbstractNumerical modules. Welke wordt weergeven hangt ten eerste af of de module alarmerend is, ofwel de waarde boven de max of onder de min. Ten tweede wordt gekeken naar de prioriteit die kan worden ingesteld bij elke AbstractNumerical module. In afbeelding .. staat ter verduidelijking de mockup.

Om dit te kunnen bouwen is er nog een nieuwe module nodig. Deze module noem ik de VitalParameter module. In de klasse van de VitalParameter module moet ik er achter komen welke AbstractNumerical module alarmerend is en de hoogste prioriteit. Daarvoor moet ik allereerst een lijst met alle modules hebben. Gelukkig heeft elke module kennis van de ModuleCollection waar het zich in bevind. Deze staat opgeslagen als attribuut in de module. In de ModuleCollection klasse heb ik een nieuwe generator methode aangemaakt die modules van het type AbstractNumerical teruggeeft. In de VitalParameter klasse itereer ik met behulp van de generator function door de modules heen. Elke module die alarmerend is wordt in een array gezet. Vervolgens sorteer ik de lijst en wordt de eerste module uit de lijst gekozen. De array met het resultaat van het algoritme wordt meegegeven aan de template, die identiek is aan die van de AbstractNumerical module.

AFBEELDING 22: DE MOCKUP VAN HET DASHBOARD OP CENTRAAL NIVEAU.

7.4 Test

De fase zijn net als bij de vorige fase unit tests gebouwd. Tijdens de uitvoer bleken alle test succesvol voltooid te zijn.

Deze fase zijn er twee nieuwe use cases gedefinieerd. Dit betekend uiteraard twee nieuwe testcases. Deze en alle andere testcases zijn terug te vinden in bijlage.

7.5 Review

Deze fase heb ik het tweede onderdeel van het project gemaakt. Hiervoor waren de plannen nog niet concreet en dus hebben we tijdens deze fase een keuze gemaakt. Gelukkig begint de rest van het Thalea project nu ook op gang te komen. NewCompliance was tot dusver nog met meerdere project bezig en Thalea had daarin niet de hoogste prioriteit. Nu is een ander project afgerond en komt de deadline voor Thalea steeds dichter in de buurt en dus is aandacht nu geheel op Thalea gezet. Ik had hier profijt van doordat er nu door de softwareanalist mockups worden gemaakt. Hierdoor wordt een veel duidelijker beeld geschapen over het gewenste eindresultaat. De opgeleverde twee modules zijn qua uiterlijk ook nagenoeg exact gelijk aan de mockups.

Tijdens de reviewsessie was iedereen erg positief. Er waren alleen wel twijfels over de locatie waar de algoritmebouwer van de AbstractNumerical modules is geplaatst. Het betreft de beheerpagina waar in principe alleen werknemers van NewCompliance toegang tot hebben. Voor de demo is deze oplossing voorlopig voldoende. In de toekomst is een extra laag tussen het beheer van NewCompliance en de dashboards gewenst. Zodat er een eigen plek is voor eindgebruikers met extra rechten. In deze laag zal dan ook de algoritmebouwer voor de AbstractNumerical modules geplaatst worden.

8 Inception fase

Tijdens deze fase heb ik het project afgerond. Ik heb hier voorafgaand geen planning gemaakt. Ik zal hieronder chronologisch mijn werkzaamheden bespreken. Ik eindig dit hoofdstuk met een aantal screenshots van het opgeleverde eindresultaat.

8.1 Uitvoering

Ik ben de fase begonnen door voor de laatste keer alle unit tests uit te voeren. Vervolgens door alle use case testgevallen uit te voeren. Deze test zijn allemaal succesvol voltooid. Toch had ik niet het idee dat op deze manier het systeem voldoende is getest. Ik heb toen nagedacht welke mogelijke testontwerptechnieken ik nog zou kunnen gebruiken. Maar geen enkele techniek zou interessante testcases opleveren. Ik heb toen besloten om het te houden op exploratory testing. Dit betekend dat ik het systeem min of meer ga misbruiken om fouten te forceren.

Om dit gemakkelijker te maken heb ik een klein programmaatje gebouwd dat de database vult met data. Zo kan ik het systeem in werking zien met een grote hoeveelheid data.

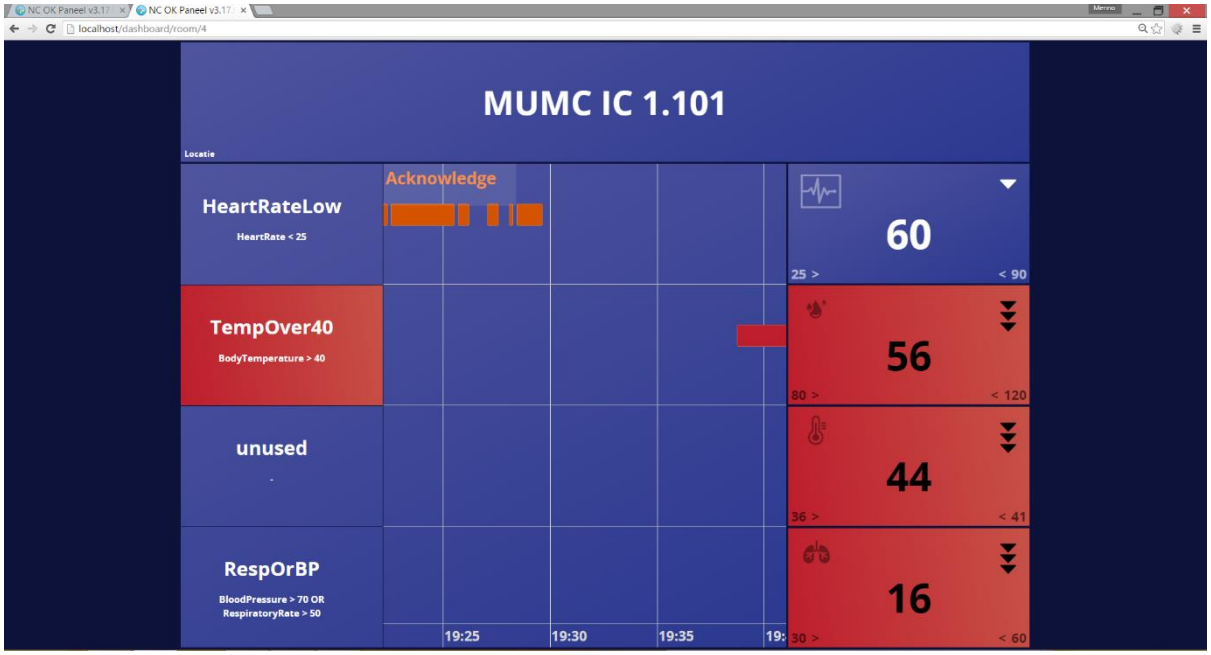
Uiteindelijk heb ik een paar fouten ontdekt die redelijk snel te verhelpen waren.

Na het testen van het systeem ben ik begonnen met het op orde stellen van de code. Zorgen dat alle indentingen goed staan en dat alle functies en methodes gedocumenteerd zijn. Vervolgens heb ik alles gecommit en gepushed naar de github repository.

Toen dacht ik klaar te zijn. Maar vervolgens werd ik gevraagd om mee te helpen met het maken van de dashboards. Aan het eind van de week zouden 2 collega’s naar Oulu in Finland gaan om een demo te geven aan een van de participerende ziekenhuizen van Thalea. De dashboards moesten voor die tijd zo ver mogelijk af zijn.

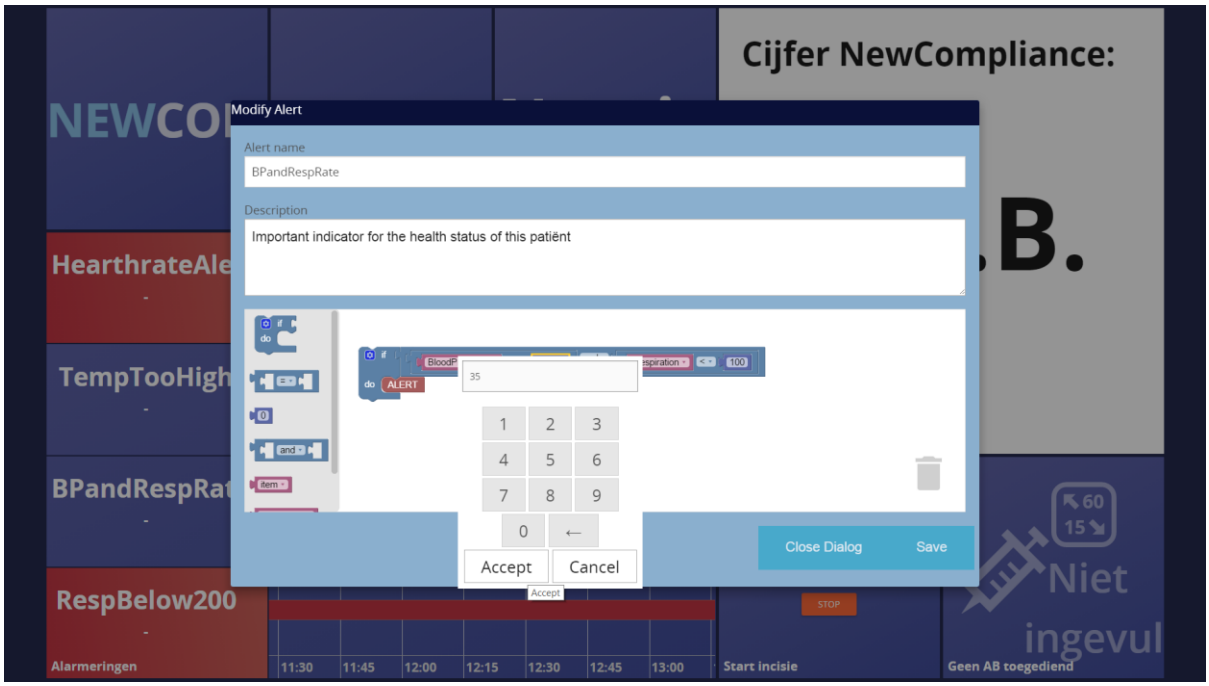
Na het opbouwen van het dashboards kwamen er toch nog een paar kleine bugs aan het licht. Maar ook deze waren snel verholpen. Vervolgens heb ik de allerlaatste commit en push uitgevoerd en dit betekende het einde van de stage.

8.2 screenshots eindresultaat

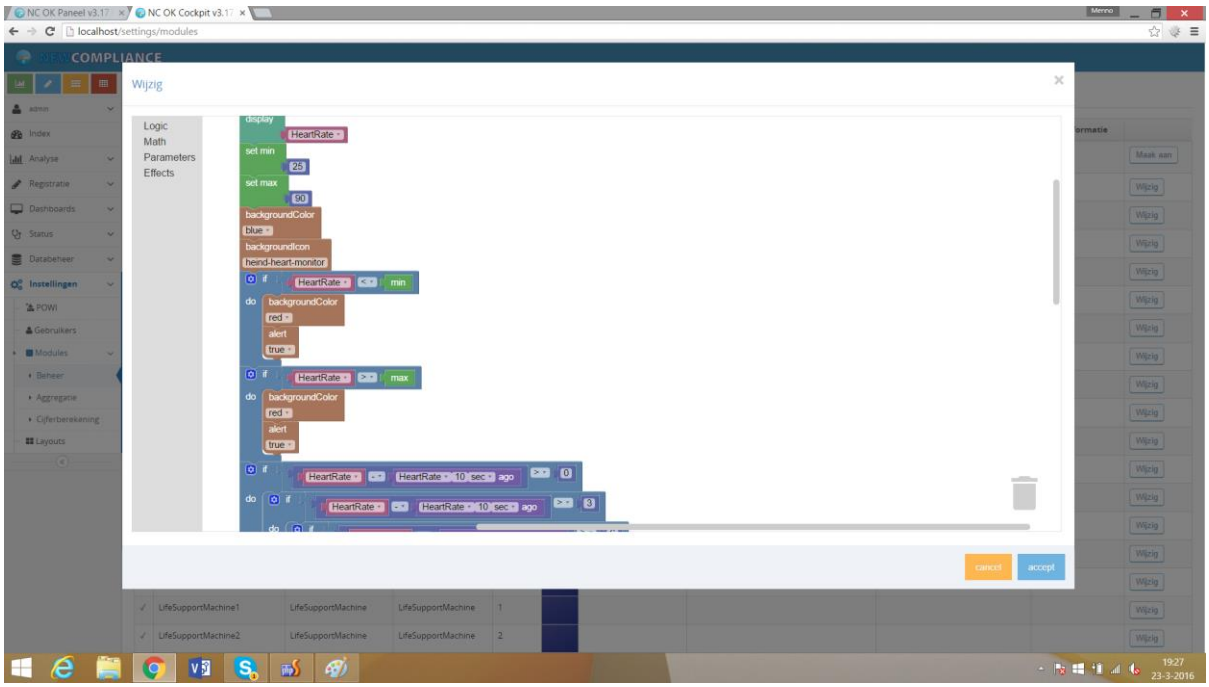


AFBEELDING 23: DE TIJDLIJNMODULE IN EEN PATIENT DASHBOARD

AFBEELDING 24: DE VITAL PARAMETER MODULES IN HET DASHBOARD OP CENTRAAL NIVEAU



AFBEELDING 25: DE ALGORITMEBOUWER MET VIRTUEEL KEYBOARD



AFBEELDING 26: DE ALGORITMEBOUWER IN MODULEBEHEER

9 Zelfreflectie

9.1 proces

Ik heb voor dit project gekozen voor de softwareontwikkelmethode openUP. De reden hiervoor was dat het de voordelen van een agile ontwikkelmethode zoals scrum combineert met de leidraad van openUP. In met aanpak is terug te zien dat ik in het project behoorlijk agile te werk ben gegaan. De documentatie die ik heb gemaakt is erg beknopt. Wellicht was een iets meer uitgebreide documentatie beter bij openUP gepast. Ik ben hier echter niet ontevreden over omdat deze aanpak niet botst met de aanpak van NewCompliance. Bij NewCompliance wordt namelijk ook agile gewerkt. Dit betekend dat de beschikbare documentie ook erg beperkt is. Dit maakt het lastig om zelf wel hele complete documentatie op te zetten. Omdat ik mee bouw aan een reeds bestaand stuk software zou ik in de documentatie graag verwijzingen maken naar onderdelen die al beschreven zijn, iets wat door de beperkte documentatie niet mogelijk is. Misschien was het gebruik van SCRUM achteraf toch een beter keuze geweest maar dit zullen we uiteraard nooit zeker weten.

Het project was opgedeeld in een 5 tal fases. De eerste fase bestond vooral uit het opzetten van het project en het kennismaken met de bestaande software. Daarnaast stond voorheen een onderzoek op de planning. Al gauw bleek dat een onderzoek overdreven zou zijn en dat een deskresearch meer dan voldoende inzicht zou geven. Aan de hand van de deskresearch is gekozen voor het framework voor visueel programmeren Blocky. Met deze keuze ben ik nog steeds dik tevreden. Het is makkelijk in gebruik en makkelijk uit te breiden.

De elaboration fase bestond uit het definiëren van de eisen en wensen, het maken van use cases en mockups. Hier heeft erg veel tijd in gezeten. Dit kwam vooral doordat de rest van NewCompliance nog met andere projecten bezig was buiten Thalea. In deze fase moest ik achteraf iets actiever achter de mensen aanzitten zodat sneller duidelijk zou zijn wat er gemaakt moest worden. Doordat ik min of meer voorliep op de rest van Thalea heb ik wel veel invloed gehad op de keuzes, zoals die voor het gebruiken van een tijdlijn om historische alarmeringen te weergeven. Het technisch ontwerp vond ik een moeilijk onderwerp in dit project. Het maken van een technisch ontwerp voor een webapplicatie met meerdere frameworks is voor zover ik kan herinneren niet behandeld op school. Daarnaast heb ik te maken met een zeer grote bestaande codebase waarvan de documentatie ook minimaal is. Het technisch ontwerp bestond hierdoor vooral uit tekeningen in mijn kladblok of op de flip-over en is daarnaast veelal mondeling besproken.

De drie construction fases verliepen grotendeels vlekkenloos en ik was daardoor ook vaak tijd over om een extra taak op te nemen. Ik heb veel kennis opgedaan over angularJS. Dit framework werd al veel gebruikt in de OK cockpit en ik heb ook veel gebruik gemaakt van het framework.

9.2 product

Ik heb uiteindelijk 3 modules opgeleverd:

- De AlertsTimeline module, of kort de tijdlijn module. Dit is verreweg de meest complexe module van de 3. De stijl past goed bij de rest van het dashboard waardoor de rustige aanblik van het dashboard behouden wordt. Het gebruik van een tijdlijn in plaats van een lijngrafiek of tekstuele logging bevorderd ook deze rustige uitstraling. De kracht van de module zit ook in de simpelheid. Dit is direct terug te zien in de hoeveelheid use-cases. Dit zijn er slechts drie. Dit komt doordat alle informatie beknopt en visueel wordt weergeven waardoor er weinig door de applicatie genavigeerd hoeft te worden.
- De AbstractNumerical. Deze module vervangt de modules voor de vitale parameters. De module kan zo aangepast worden dat hij de vorm kan aannemen van elke vitale parameters module. Dit bied de gebruiker de mogelijkheid om de modules geheel naar zijn wens aan te passen. Ik ben erg tevreden over het resultaat. De module ziet er uit zoals in de mockups is voorgesteld. Maar zoals als eerder verteld is de locatie in de software waar het algoritme voor de module gebouwd kan worden niet ideaal.
- De VitalParameters module. Deze module is een uitbreiding op de AbstractNumerical module. Het zal de AbstractNumerical met de hoogste prioriteit die alarmerend is weergeven. De vraag voor deze module komt direct uit een mockup van de softwareanalist. Het doet exact wat gevraagd is en dus ben ik tevreden.

In zijn geheel ben ik en is ook de rest van NewCompliance er tevreden. Wat ik zelf gaaf vind om te zien is dat de door mij gemaakte modules de belangrijkste zijn voor het Thalea project. De dashboards met mijn modules zijn in een demo aan klanten laten zien in Finland. In de afbeelding hieronder is het dashboard dat de VitalParameters module gebruikt werkend te zien op een scherm in het ziekenhuis van Oulu in Finland.

AFBEELDING 27: WERKENDE DASHBOARD MET DE VITALPARAMETER MODULES TIJDENS DE DEMO IN OULU

9.3 beroepstaken

3.2 Ontwerpen systeemdeel

Vanuit een technisch oogpunt was dit een lastig onderdeel. Vanuit NewCompliance is er weinig documentatie beschikbaar. Hierdoor is het lastig om een technisch ontwerp te maken dat voortbouwt op bestaande code. Hierdoor bestond het technisch ontwerp vooral uit tekeningen in een kladblok of flipover. Dit heeft ook te maken met de softwareontwikkelmethode die NewCompliance hanteert. Mondeling contact wordt belangrijker gevonden dan documentatie. Ik ben daardoor ook vaak in gesprek geweest met de softwarearchitect om de aanpak te bespreken.

Aan de functionele kant heb ik vooraf altijd mockups gemaakt. Deze mockups zeggen veel over de hoe ik de eisen en wensen heb opgevat. Mocht dit plaatje niet kloppen dan kan er optijd bijgestuurd worden. De mockups heb ik voor het grootste deel gemaakt door verschillende afbeeldingen aan elkaar te knippen en plakken. De interface is van groot belang bij NewCompliance. De rustige interface van de OK cockpit waarbij zo veel mogelijk tekst wordt vermeden is uniek in de medische wereld. Het was daarom belangrijk om hier rekening mee te houden. Ik denk daar ik daar goed aan heb voldaan. De modules geven veel informatie op zo'n efficiënt mogelijke manier met zo min mogelijk tekst weer.

3.3 Bouwen applicatie

Ik heb tijdens de afstudeeropdracht verder gebouwd aan bestaande software. Ik heb hierbij rekening gehouden met hergebruik van de code. Dit is direct terug te zien doordat de interpreter meerdere keren in de software gebruikt wordt. Ik heb de flexibiliteit van de taal PHP goed benut door bijvoorbeeld het gebruik van variabele functies. Naast PHP heb ik gebruik gemaakt van een hele reeks andere programmeertalen en frameworks. Daarvan zijn de belangrijkste: Laravel, AngularJS en Javascript.

Ik heb de versiebeheertool git gebruikt om de code veilig op te slaan en om de mogelijkheid te hebben terug te keren naar een vorige versie van de code.

3.4 Initieren en plannen van het testprocess

Ik heb de functionaliteiten van het systeem getest door use case tests uit te voeren. Deze testgevallen ontstaan uit de gedefinieerd use cases. Voor main flow en alternatieve flow van de use case beschrijvingen heb ik een testgeval gemaakt.

Tijdens het bouwen heb ik het systeem getest door middel van unit tests. Ik heb hiervoor het framework PHPunit gebruikt. Met unit tests test je onderdelen van het systeem zonder dat deze afhankelijk zijn van de rest van het systeem. Indien voor het uitvoeren van een onder toch een ander onderdeel nodig was heb ik een mock object gemaakt om dit onderdeel te vervangen.

Bibliografie

Green, T. (1995). *Programming plans, imagery, and visual programming.* INTERACT.

Green, T., & Petre, M. (1996). *Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework.*

Pandey, R., & Burnett, M. (1993). *Is it easier to write matrix manipulation programs visually or textually? An empirical study.*

Soloway, E., Ehrlich, K., Bonar, J., & Greenspan, J. (1984). *What do novice know about programming?*

Lijst van bijlagen

Bijlage A

Afstudeerplan

Informatie afstudeerder en gastbedrijf (*structuur niet wijzigen*)

Afstudeerblok: 2016-1.1 (start uiterlijk 8 februari 2016)
Startdatum uitvoering afstudeeropdracht: 30 november 2015
Inleverdatum afstudeerdossier volgens jaarrooster: 3 juni 2016

Studentnummer: 09013342
Achternaam: dhr Postma
Voorletters: M.T.
Roepnaam: Menno
Adres: Hazerswoudestraat 16
Postcode: 2729DA
Woonplaats: Zoetermeer
Telefoonnummer: 079-3422561
Mobiel nummer: 06-11773879
Privé emailadres: mennodanmaar@hotmail.com

Opleiding: Informatica
Locatie: Zoetermeer
Variant: voltijd

Naam studieloopbaanbegeleider: A.A.A.M Jacobs
Naam begeleidend examinator: A.A.A.M Jacobs
Naam tweede examinator: T.K. Cocx

Naam bedrijf: NewCompliance
Afdeling bedrijf: Softwareontwikkeling
Bezoekadres bedrijf: Cobaltstraat 26
Postcode bezoekadres: 2718 RM
Postbusnummer:
Postcode postbusnummer:
Plaats: Zoetermeer
Telefoon bedrijf: +31 (0)79-7370000
Telefax bedrijf: +31 (0) 84 8354228
Internetsite bedrijf: http://newcompliance.nl/

Achternaam opdrachtgever: dhr van Lamsweerde
Voorletters opdrachtgever: G
Titulatuur opdrachtgever: MSc
Functie opdrachtgever: Product Manager
Doorkiesnummer opdrachtgever: -
Email opdrachtgever: g.vanlamsweerde@newcompliance.nl

Achternaam bedrijfsmentor: dhr Koene
Voorletters bedrijfsmentor: M
Titulatuur bedrijfsmentor: BSc
Functie bedrijfsmentor: Project Leader
Doorkiesnummer bedrijfsmentor: -
Email bedrijfsmentor: m.koene@newcompliance.nl

Doorkiesnummer afstudeerder: -
Functie afstudeerder (deeltijd/duaal): Voltijd Stagiair Softwareontwikkelaar

Titel afstudeeropdracht: Ontwikkelen van een gebruikersvriendelijk stuk software voor het maken en aanpassen van algoritmes bij NewCompliance.

Opdrachtomschrijving

1. Bedrijf

NewCompliance is een jong commercieel bedrijf gevestigd in Zoetermeer. Opgericht in 2006 als spin-off bedrijf van TU-Delft. NewCompliance ontwikkeld innovatieve producten, gericht op het verbeteren van patiëntveiligheid en productiviteit. De markt waar NewCompliance op richt is dat van ziekenhuizen. Een markt waar nog grote stappen op het gebied van IT gezet kunnen worden. Het bedrijf bestaat uit twee afdelingen; softwareontwikkeling en sales. De afstudeeropdracht zal worden uitgevoerd in de afdeling softwareontwikkeling. Hier worden de innovatieve producten ontwikkeld en beheerd. De sales afdeling is verantwoordelijk voor klanten vergaren en verkoop van producten. Naast het verkopen van de door NewCompliance ontwikkelde oplossingen is NewCompliance ook de enige Nederlandse distributeur van het Vernacare® systeem. NewCompliance heeft op dit moment 14 full-time medewerkers en is momenteel sterk aan het groeien. Het belangrijkste softwareproduct is de OK cockpit. Een dashboard waarop belangrijke informatie tijdens een operatie in de operatiekamer wordt weergeven. Het systeem geeft waarschuwingen gebaseerd op actuele gegevens en aan het eind van een operatie word een rapport opgesteld dat veel zegt over de veiligheid van de patiënt.

2. Probleemstelling

NewCompliance neemt deel aan het internationale Thalea project. Dit is een samenwerkingsverband tussen meerdere ziekenhuizen en bedrijven. Er is steeds meer bewijs dat telemedicin in de intensive care levensreddend kan zijn. Telemedicin is een verzamelnaam voor IT oplossingen in de geneeskunde. CORDIS, onderdeel van de Europese Commissie heeft vastgesteld dat een telemedicin-platform met een hoge interoperabiliteit dat onafhankelijk is van de leverancier nog niet bestaat, terwijl dit wel gewenst is. Hiervoor is het Thalea project in het leven geroepen. Meerdere bedrijven zullen een oplossing bedenken en een prototype bouwen. Vervolgens zullen de beste oplossingen gekozen worden om door te gaan naar de volgende fase; het bouwen van het in productie te nemen product.

De hoge interoperabiliteit van systeem moet behaald worden door verschillende systemen in de ziekenhuizen met elkaar te laten communiceren en samenwerken. Dit zal de kenniskloof in Europa tussen ziekenhuizen verkleinen en zo doctors helpen meer patiënten te genezen zodat meer patiënten onafhankelijk thuis kunnen wonen.

Het systeem moet leverancier onafhankelijk zijn. Dit vraagt om een zeer flexibel systeem waarvan de werking in grote mate aanpasbaar is. Het systeem zal bestaan uit een dashboard waarop informatie over de patiënt wordt weergegeven. Vergelijkbaar met de dashboard van de OK-cockpit zoals beschreven onder “bedrijf”. Echter wordt de informatie ditmaal aangevuld met informatie uit andere ziekenhuizen.
Wélke informatie, waar, wanneer en hoe wordt weergeven op het dashboard, wordt bepaald door een grote hoeveelheid algoritmes. Om het systeem flexibel te maken moeten deze algoritmes voor de eindgebruiker aanpasbaar zijn. Het realiseren van een algoritme-systeem die het maken en aanpassen van algoritmes faciliteert is het doel van de afstudeeropdracht.

3. Doelstelling van de afstudeeropdracht

Het nieuwe systeem voor het Thalea project wordt gebaseerd op het bestaande OK-cockpit systeem. Het algoritme-systeem zal eerst gebouwd worden als aansluiting op de OK-cockpit. Zodra de bouw van het Thalea-systeem ver genoeg is gevorderd kan het algoritme systeem daar gemakkelijk op worden aangesloten.

Het algoritme-systeem moet het voor de eindgebruiker mogelijk maken om zelf algoritmes te bouwen en/of aan te passen. Hoe deze doelstelling behaald wordt zal tijdens het project bedacht worden. Er moet een keuze gemaakt worden in de hoeveelheid vrijheid in het bouwen van algoritmes die de gebruiker gegeven wordt. Daarnaast heeft de technische mogelijkheid binnen de beperkte looptijd van het project invloed op de invulling van het project.

4. Resultaat

Met het algoritme-systeem wordt de eindgebruiker minder afhankelijk van de leveranciers. Daarnaast kunnen gewenste aanpassingen aan de algoritmes sneller worden doorgevoerd. Dit zal uiteindelijk leiden tot meer tevredenheid bij de klant.

Het algoritme-systeem vervult een aantal requirements met een hoge prioriteit van het Thalea project. Als het algoritme systeem is gerealiseerd brengt dit NewCompliance een stap dichter bij de volgende fase van Thalea. Dit kan uiteindelijk de internationale doorbraak van NewCompliance betekenen

5. Uit te voeren werkzaamheden, inclusief een globale fasering, mijlpalen en bijbehorende activiteiten

De (globale) planning is gebaseerd op de softwareontwikkelmethode OpenUp. Per fase zal een detailplanning gemaakt worden tijdens de voorafgaande fase.

	Dagen:
• **Inception phase (3 weken):**	**15**
○ Vision vaststellen	
○ Plan van aanpak schrijven	
○ Brainstorm sessie met stakeholder	
○ Identificeren en opstellen requirements	
○ Deskresearch	
○ Ontwikkelomgeving opzetten	
○ Testplan	
• **Elaboration phase (2 weken):**	**9**
○ Identificeren en verfijnen requirements	
○ Opstellen use-cases	
○ Testcases opstellen	
○ Technisch ontwerp	
• **Construction phase1 (3 weken):**	**13**
○ Developer tests opstellen	
○ Oplossing ontwikkelen	
○ Developer tests uitvoeren	
○ Testcases verfijnen	
○ Tests uitvoeren	
• **Construction phase2 (3 weken):**	**13**
○ Developer tests opstellen	
○ Oplossing ontwikkelen	

    - Developer tests uitvoeren
    - Testcases verfijnen
    - Tests uitvoeren
- **Construction phase3 (4 weken):** **13**
    - Developer tests opstellen
    - Oplossing ontwikkelen
    - Developer tests uitvoeren
    - Testcases verfijnen
    - Tests uitvoeren
- **Transition phase (2 week):** **6**
    - Acceptatietest uitvoeren
    - Documentatie afmaken
    - Opleveren aan NC

Overige 15 dagen: Verslaglegging afstuderen.

6. Op te leveren (tussen)producten

Dit zijn de (tussen)producten die opgeleverd zullen worden.

- Plan van aanpak
- Requirements
- Use cases
- Testplan
- Technical design
- Prototype 1
- Testrapport 1
- Prototype 2
- Testrapport 2
- Eindproduct (algoritme-systeem)
- Testrapport 3
- Handleiding

7. Te demonstreren competenties en wijze waarop

3.2 Ontwerpen systeemdeel

Het stuk software dat gebouwd moet worden moet voldoen aan een hoge gebruiksvriendelijkheid. Het maken van een interface waarmee mensen, buiten het vakgebied IT, algoritmes kunnen bouwen zorgt voor een grote uitdaging op het gebied van de GUI. Na het bouwen van een algoritme moet deze gebruikt kunnen worden gebruikt in de OK cockpit. Hoe het algoritme (bij voorkeur tijdens runtime) in de OK-cockpit terechtkomt is een erg lastig probleem waar een sterk design voor bedacht moet worden. Hiervoor zullen technieken als UML, flowcharts, PSD's en pseudo code gebruikt worden. Omdat het algoritme-systeem in zowel de bestaande OK cockpit als het Thalea-systeem geïmplementeerd moet kunnen worden is een vorm van abstractie tussen het algoritme-systeem en de rest van het systeem nodig.

3.3 Bouwen applicatie

Het product sluit aan op een bestaand systeem. Zowel de front-end als de back-end moet gebouwd worden. Hierbij kan gebruikt gemaakt worden van API's en frameworks. Gezien de opdracht zal de back-end complex worden. Er wordt verwacht dat geavanceerde concepten van de programmeertaal gebruikt zullen worden.

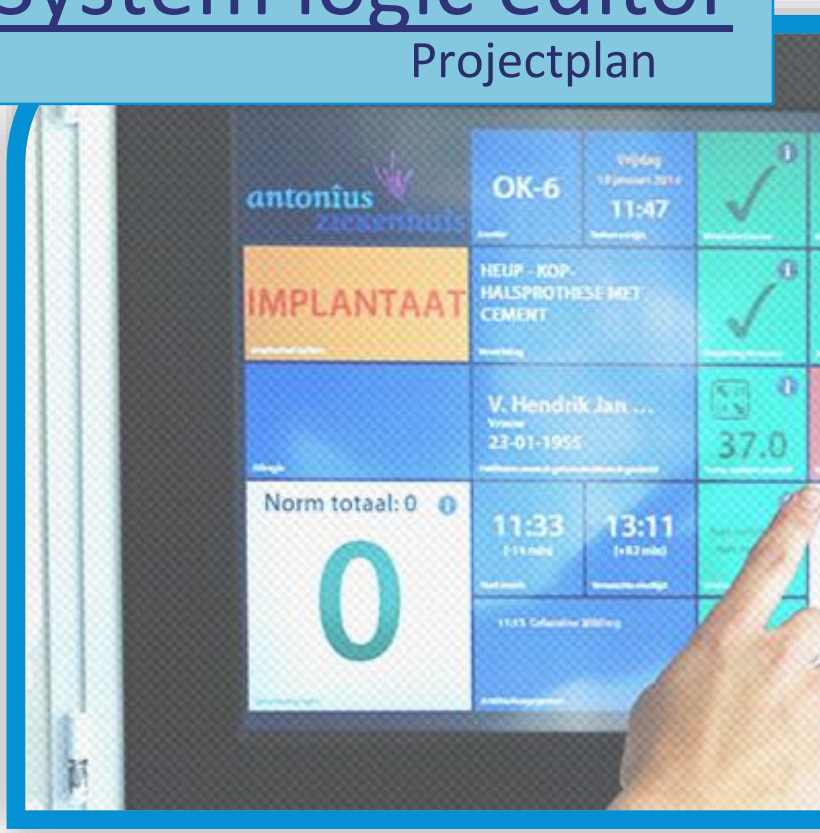
3.4 Initiëren en plannen van het testproces

Al vroeg in het project zal het testplan worden opgesteld. Zo kan de keuze gemaakt worden de tests parallel aan de ontwikkelstraat op te stellen. Uiteraard kunnen de test later in het traject aangepast en verfijnd worden. Er zal een risicoanalyse worden uitgevoerd om inzicht te krijgen waar de risicovolle onderdelen zitten. De tests kunnen op de resultaten van de testrisicoanalyse afgesteld worden.

Bijlage B

Thalea: System logic editor

Projectplan



Menno Postma, Stagiair
NewCompliance
19-11-2015, Zoetermeer
m.postma@newcomplaince.com

1 Inhoudsopgave

1 Inleiding

Dit is het plan van aanpak voor het project “System logic editor”, wat een onderdeel is van het internationale Thalea project. In het Thalea project wordt gezocht naar een platform waarmee informatie van verschillende ziekenhuizen uitgewisseld kan worden. Vervolgens moet deze informatie dienen als input voor dashboards die in IC kamers worden geïnstalleerd. Tussen de input en de weergave op de dashboards zitten complexe algoritmes die bepalen wat en wanneer informatie weergeven moet worden. Een van de wensen is de mogelijkheid om, zonder IT kennis, deze algoritmes aan te passen en nieuwe algoritmes aan te maken.

In dit document wordt besproken hoe het project uitgevoerd zal worden. In het hoofdstuk opdrachtformulering wordt de aanleiding, probleemstelling en doelstelling van het project uitgelegd. In het hoofdstuk aanpak worden keuzes gemaakt die betrekking hebben op de uitvoer van het project. Denk hierbij aan de keuzes voor de softwareontwikkelmethode en kwaliteitswaarborging. In de planning wordt eerst gekeken welke taken uitgevoerd moeten worden en hoeveel tijd deze taken zullen kosten. Vervolgens wordt aan de hand van deze informatie de planning opgesteld.

2 Opdrachtomschrijving

2.1 Aanleiding

Voor een internationaal project wordt op dit moment een nieuw product ontwikkeld genaamd IC cockpit. Het is gebaseerd op de OK cockpit maar zal toegepast worden in de Intensive care. Dit product zal net als de OK cockpit bestaan uit een dashboard met belangrijke informatie en de mogelijkheid een rapport te genereren. Wanneer een waarschuwing weergeven moet worden en hoe het rapport wordt opgebouwd wordt bepaald door complexe algoritmes in het systeem. Het is lastig te voorspellen hoe de gewenste algoritmes eruit moeten zien. Als een klant niet tevreden is met de huidige algoritmes is het de taak dat NewComliance wederom een nieuwe versie moet ontwikkelen. Dit maakt de klanten sterk afhankelijk van de leverancier. Daarnaast kost het veel tijd voordat een gewenste veranderingen aan de algoritmes daadwerkelijk bij de klant geïmplementeerd zijn.

Dit probleem vraagt om systeemdeel waarmee een persoon zonder IT kennis zelf veranderingen kan maken aan de algoritmes. De interface moet het bouwen en/of aanpassen van algoritmes op een gebruiksvriendelijke wijze mogelijk maken.

2.2 Doelstelling

Met dit component wordt de klant minder afhankelijk van de leveranciers. Daarnaast kunnen gewenste aanpassingen aan de algoritmes sneller worden doorgevoerd. Dit zal uiteindelijk leiden tot meer tevredenheid bij de klant.

NewCompliance zal minder tijd en geld aan beheer kwijt zijn, waardoor er meer middelen overblijven voor het ontwikkelen van nieuwe innovatie producten. Dit zal de al aanwezige groei bij NewCompliance nog meer ten goede komen.

2.3 Opdrachtformulering

Het systeemdeel zal ontwikkeld worden als aansluiting op de OK cockpit. Zodra de IC cockpit gerealiseerd is, kan het systeemdeel daar gemakkelijk aan gekoppeld worden.

Het te ontwikkelen systeemdeel moet het voor klanten mogelijk maken om zelf algoritmes te bouwen en/of aan te passen. Daarbij is het belangrijk goede keuzes te maken in de hoeveelheid vrijheid die de klant krijgt. Ook zal er goed nagedacht moeten worden over hoe een aangepast of nieuw algoritme door het systeem in gebruik genomen moet worden. Aangezien het systeem wordt toegepast in de medische wereld is veiligheid een belangrijk aspect. In technisch ontwerp moet hier rekening mee gehouden worden en gedurende het hele traject moet uitgebreid getest worden.

2.4 Uitgangspunten en randvoorwaarden

2.4.1 Randvoorwaarden

- Doorlooptijd: 16-11-2015 tot 21-3-2016
- Teamgrootte: 1 softwareontwikkelaar
- Programmeertalen: PHP, Javascript, CSS, HTML
- Databasesysteem: postgreSQL

- Tools en methoden: AngularJS, laravel equolent

2.4.2 Binnen scope

- Systeemdeel waarmee algoritmes kunnen worden gebouwd en aangepast
- Invoeren van benodigde algoritmes voor de eerste oplevering via het systeemdeel
- Aansluiting op OK cockpit
- Aansluting op IC cockpit
- Handleiding opstellen voor de eindgebruiker
- Handleiding opstellen voor installatie en uitleg over de werking

2.4.3 Buiten scope

- Training voor de eindgebruiker
- Beheer van het eindproduct

3 Projectorganisatie

Beschrijving van de organisatiestructuur van het project.

3.1 Projectleden en rollen

Persoon	Rollen	Verantwoordelijkheid
Guillaume van Lamsweerde	Analyst	System wide requirements, vision
Sylvester Oosterbos	Architect	Architecture, Code review
Menno Postma	Project Manager, Developer, Tester	Project plan, iteration plan, design, build, tests

3.2 Communicatie

De communicatie zal grotendeels mondeling gebeuren. Afspraken worden gemaakt via google agenda. Via trello wordt de voortgang van het project bijgehouden. Op trello kunnen de projectleden relevante documenten uploaden en reageren op het process. Communicatie kan daarnaast via slack, mail of telefoon.

- Trello: https://trello.com/b/uZfBa7xP/afstuderen-menno-postma (account en invite vereist)
- Slack: https://newcompliance.slack.com (account en invite vereist)
-

Persoon	E-mail	Telefoon
Guillaume van Lamsweerde	g.vanlamsweerde@newcompliance	
Sylvester Oosterbos	s.oosterbos@newcompliance.nl	
Menno Postma	m.postma@newcompliance.nl	

4 Projectplanning

In de projectplanning wordt een plan gemaakt voor de uitvoering van het project. Allereerst zal gekozen worden voor een softwareontikkelmethode. Aan de hand van deze methode zal het project verder worden ingepland.

4.1 Softwareontwikkelmethode

Het doel van het project is duidelijk. Hoe dit doel wordt bereikt is nog niet verder gespecificeerd. Er zijn verschillende mogelijkheden om het doel te bereiken. Om hier een goede keuze in te maken is het een goed idee om gebruik te maken van prototyping. Er wordt dan vroeg in het project een uitgeklede versie van het eindproduct opgeleverd. Hier kan vervolgens uitgebreid feedback op gegeven worden. Indien nodig kan het project dan tijdig worden bijgestuurd. Het risico, dat het product niet aan de wensen van de klant voldoet, neemt hiermee af. Er zijn meerder softwareontwikkelmethoden waar prototyping in toegepast kan worden. We praten hier vooral over iteratieve methoden

Twee methodes waar naar gekeken is zijn SCRUM en RUP. Tussen deze twee methodes zijn een aantal overeenkomst en verschillen, wat voor en nadelen ten opzicht van elkaar met zich meebrengt.

SCRUM Kenmerken	**RUP Kenmerken**
Iteratief	Iteratief
Snel ontwikkelen	Meer gedocumenteerd
Minimalistisch. Meer een manier van werken dan een volledige ontwikkelmethode	Uitgebreid. Meer dan een manier van werken. Ondersteund het project. Maar is mischien te uitgebreid voor dit relatief kleine project
Voegt weinig toe voor solo project	
Meer tijd over voor ontwikkelen	Meer tijd kwijt aan project management
Conform agile manifesto	

De grootste verschillen tussen beide methodes zitten in tijd en documentatie. Doordat RUP veel documentatie vraagt van de gebruiker, is de gebruiker veel tijd kwijt. Daarentegen bied RUP meer hulp en leidraad in het in de juiste banen geleiden van het project. Er zijn progingen gedaan de nadelen van RUP te verminderen. Een van deze progingen is geresulteerd in OpenUP. Een methode die het beste van de twee werelden, RUP en SCRUM samenvoegd. Zo is een ontwikkelmethode ontstaan die de onderstuening bied van RUP met de voordelen van een pure agile methode zoals SCRUM.

4.2 Globale iteratiedoelen

De elaboration en contruction phase kunnen geïterreerd worden. Hoeveel iteraties hangt af van het project. Voor ons probleem zijn meerdere oplossing mogelijk, daarom is gekozen voor twee elaboration phases. Dit bied de mogelijkheid om twee verschillende oplossingen te ontwerpen, en daar feedback op ontvangen, voordat wordt begonnen met de bouw. Het product zal incrementeel gebouwd worden. Dit betekend dat al snel een werkende versie wordt opgeleverd die vervolgens steed wordt bijgewerkt. Er is gekozen voor 3 construction phases zodat er in ieder geval 2 protoypes worden opgeleverd en gereviewd.

Fase	#	Periode	Doelen
Inception	1	16/11/2015 t/m 27/11/2015	➢ Opdracht formuleren ➢ Planning maken ➢ Requirements achterhalen ➢ Use cases opstellen
Elaboration	1	30/11/2015 t/m 04/12/2015	➢ Technisch ontwerp maken ➢ GUI ontwerpen ➢ Requirements verfijnen
	2	07/12/2015 t/m 11/12/2015	➢ Technisch ontwerp verfijnen of vernieuwen ➢ GUI verfijnen of vernieuwen ➢ Requirements verfijnen
Construction	1	14/12/2015 t/m 08/01/2016	➢ Bouw prototype
	2	11/01/2016 t/m 12/02/2016	➢ Bouw prototype
	3	15/02/2016 t/m 04/03/2016	➢ Bouw prototype
Transition	1	14/03/2016 t/m 18/03/2016	➢ Oplevering eindproduct ➢ Handleidingen opstellen

4.3 Opleveringen

Er zullen in het project 3 opleveringen zijn.

- Prototype 1: Dit zal een werkend systeemdeel betreffen die gekoppeld is aan een versie van OK cockpit. Het zal een uitgeklede versie zijn van de uiteindelijke doelstelling. Developer tests zullen uitgevoerd worden en andere tests die relevant zijn. Dit zal afhangen van de staat van het prototype.
- Prototype 2: Idem ditto aan prototype 1. Uiteraard zal het systeemdeel op dit moment wel een stuk volledige zijn.
- Eindproduct: Het eindproduct moet voldoen aan alle functionele en niet-functionele eisen voldoen. Zowel de developer tests en acceptatie tests worden uitgevoerd. Deze versie word bij voorkeur aan de IC cockpit gekoppeld. Is dit nog niet mogelijk dan zal hij aan de OK cockpit gekoppeld worden.

4.4 Projectagenda

Fase	Artifact	Deadline
Inception	Project plan	20-11-2015
	Test plan	27-11-2015
Elaboration	Requirements	4-12-2015
	Use case	4-12-2015
	Design	11-12-2015
	Functional test cases	11-12-2015
Construction	Prototype	8-1-2015 en 12-2-2015
	Developer tests	4-3-2015
Transition	Eindproduct	11-3-2015
	Handleiding	18-3-2015

4.5 Planning

De detailplanning zal per iteratie gemaak worden. Dit zal gedaan worden in de tool trello. De planning is fexibel doordat het product incrementeel ontwikkeld zal worden. Wel zijn er een aantal mijlpalen die een vaste deadline hebben zoals beschreven in de project agenda. Deze deadlines voorkomen dat er te veel van de globale planning afgeweken wordt.

5 Projectbewaking

5.1 Scopebewaking

5.2 Kwaliteitbewaking

- Code conventie zoals: http://wiki.newcompliance.nl/Conventies
- Wekelijkse code review

5.3 Risicobewaking

5.3.1 Risico's

1. De doelstelling veranderd gedurende het project.
2. Het Thalea project, of de deelname van NewCompliance aan het Thalea project, wordt opgeheven.
3. De gekozen oplossing is technisch niet haalbaar binnen het project
4. De IC cockpit wijkt dermate van de OK cockpit af dat de aansluiting op de IC cockpit meer tijd kost dan verwacht

5.3.2 Maatregelen

1. Meerdere iteraties met uitgebriede review sessie zodat veranderingen in de wensen optijd geïndentificeerd worden.
2. In dit geval zal het project gewoon doorlopen. Het product zal in dit geval op de OK cockpit aangesloten worden tijdens de oplevering
3. Een uitgebreide elaboration fase met meerdere iteraties zal zorgen voor een backup plan.
4. De bouw van de IC cockpit zal gevolgd worden door aanwezig te zijn bij de besprekingen over Thalea. Het risico kan op deze manier snel herkend worden zodat er passende maatregelen genomen kunnen worden.

Bijlage C

Requirements document

Wijzigingen

Datum	wat	welke
5-1-2016	Requirements toegevoegd	FR5, FR6
5-1-2016	Use case toegevoegd	UC3
16-2-2016	Requirements toegevoegd	FR4a, FR4b, FR4c, FR4d, FR4e
16-2-2016	Use cases toegevoegd	UC4, UC5
7-3-2016	Requirement toegevoegd	FR7
7-3-2016	Use cases toegevoegd	UC6, UC7, UC8

Requirements

ID	Requirement
FR1	De gebruiker moet alarmeringen kunnen instellen in het dashboard op patiënt niveau.
FR2	Het moet mogelijk zijn een alarm te bouwen aan de hand van een threshold, trend of een combinatie van beide.
FR3	De gebruiker moet naast huidige alarmeringen ook de historische alarmeringen kunnen zien
FR4	De beheerder moet de logica van een nieuw type module grotendeels kunnen bepalen (verder specificeren)
FR4a	De beheerder kan bepalen onder welke condities de achtergrondkleur van de module verandert
FR4b	De beheerder kan bepalen onder welke condities de achtergrond icoon van de module verandert
FR4c	De beheerder kan bepalen onder welke condities de richting van de pijltjes van de module verandert
FR4d	De beheerder kan bepalen onder welke condities de kleur van de pijltjes van de module verandert
FR4e	De beheerder kan bepalen welke waarde in de module wordt weergegeven
FR4f	De beheerder kan bepalen onder welke condities de module alarmerend is (wordt gebruikt door vitale parameters module)
FR5	Kleur een alarm rood wanner deze alarmerend is.
FR6	Kleur een alarm geel wanneer er een alarmering is geweest totdat de gebruiker aangeeft het alarm gezien te hebben.
FR7	Een nieuwe type module weergeeft de abstract numerieke module die alarmerend is en de hoogste prioriteit heeft.

Use case list

Use case ID	Use Case	Beschrijving	Requirement ID
UC1	Bekijk tijdlijn module	De gebruiker ziet informatie over de ingestelde alarmen	FR3, FR5, FR6
UC2	Wijzig alarm	De gebruiker wijzigt een alarm	FR1, FR2
UC3	Acknowledge alarmering	De gebruiker geeft aan dat de alarmering gezien is.	FR6
UC4	Bekijk abstracte numerieke module	De gebruiker ziet informatie over de ingestelde module	FR4
UC5	Wijzig abstracte numerieke algoritme	De gebruiker wijzigt een abstracte numerieke module	FR4
UC6	Bekijk vitale parameter module	De gebruiker ziet informatie van de abstract numerieke modules met de hoogste prioriteit	FR7
UC7	Stel prioriteit voor abstracte numerieke module in	De gebruiker stelt een prioriteit in voor de abstracte numerieke module	FR7
UC8	Stel prioriteit voor vitale parameter module in	De gebruiker stelt de prioriteit in die bepaald welke abstracte numerieke module getoond wordt	FR7

Use case beschrijvingen

ID	UC1
Naam	Bekijk tijdlijn module
Beschrijving	De gebruiker ziet informatie over de ingestelde alarmen
Actoren	Gebruiker
Precondities	Een dashboard dat de alerts timeline module implementeert is geopend.
Verloop	1. Het systeem toont de tijdlijn module
Postcondities	-
Alternatief verloop	-

ID	UC2
Naam	Wijzig alarm
Beschrijving	De gebruiker wijzigt een alarm
Actoren	Gebruiker
Precondities	Een dashboard dat de alerts timeline module implementeert is geopend.
Verloop	1. De gebruiker klikt op een alarm 2. Een formulier voor naam, beschrijving en algoritme wordt geopend 3. De gebruiker vult de benodigde informatie in 4. De gebruiker klikt op de knop voor opslaan.
Postcondities	De aanpassingen aan de alert zijn opgeslagen in de database.
Alternatief verloop	• Na stap 2 klikt de gebruiker op de annuleer knop. Het formulier wordt gesloten en de alert wordt niet gewijzigd/opgeslagen.

ID	UC3
Naam	Acknowledge alarmering
Beschrijving	De gebruiker geeft aan dat de alarmering gezien is.
Actoren	Gebruiker
Precondities	• Een dashboard dat de alerts timeline module implementeert is geopend. • In de database staat een actieve alarmering
Verloop	1. Het systeem toont de tijdlijn module 2. Het systeem detecteert dat een alarmering actief is 3. Het systeem kleurt de tijdlijn geel 4. Het systeem weergeeft de acknowledge knop 5. De actor klikt op de acknowledge knop 6. Het systeem kleurt de tijdlijn terug naar de oorspronkelijke kleur
Postcondities	-
Alternatief verloop	-

ID	UC4
Naam	Bekijk abstracte numerieke module
Beschrijving	De gebruiker ziet informatie over de ingestelde module
Actoren	Gebruiker
Precondities	Een dashboard dat de abstracte numerieke module implementeert is geopend.
Verloop	1. Het systeem toont de abstracte numerieke module
Postcondities	-
Alternatief verloop	-

ID	UC5
Naam	Wijzig abstracte numerieke algoritme
Beschrijving	De gebruiker wijzigt een abstracte numerieke module
Actoren	Gebruiker
Precondities	De modulebeheer pagina is geopend
Verloop	1. Het systeem toont de lijst met alle modules 2. De gebruiker klikt op de wijzig knop van een abstracte numerieke module 3. Het systeem opent een formulier 4. De gebruiker voert informatie in de velden in 5. De gebruiker klikt op de knop opslaan
Postcondities	De gewijzigde module is opgeslagen in de database
Alternatief verloop	1. Bij stap 4 klikt de gebruiker op de wijzig algoritme knop 2. Het systeem opent een venster waar een algoritme gebouwd kan worden 3. De gebruiker klikt op opslaan 4. Het venster wordt gesloten en het algoritme staat in het veld ingevuld 5. Door naar 5 1. Na stap 2 van het alternatieve verloop. 2. De gebruiker klikt op annuleren. 3. Het algoritme wordt verworpen 4. Het venster wordt gesloten. 5. Het algoritme in het veld is ongewijzigd

ID	UC6
Naam	Bekijk vitale parameter module
Beschrijving	De gebruiker ziet informatie van de abstract numerieke modules met de hoogste prioriteit
Actoren	Gebruiker
Precondities	Een dashboard dat de vitale parameter module implementeert is geopend.
Verloop	1. Het systeem toont de vitale parameter module
Postcondities	-
Alternatief verloop	-

ID	UC7
Naam	Stel prioriteit voor abstracte numerieke module in
Beschrijving	De gebruiker stelt een prioriteit in voor de abstracte numerieke module
Actoren	Gebruiker
Precondities	De modulebeheer pagina is geopend
Verloop	1. Het systeem toont de lijst met alle modules 2. De gebruiker klikt op de wijzig knop van een abstracte numerieke module 3. Het systeem opent een formulier 4. De gebruiker breid in het opties veld het json object uit met de waarde ‘priority’ met een geheel getal 5. De gebruiker klikt op opslaan
Postcondities	De prioriteit is opgeslagen bij de module in de database onder het opties veld.
Alternatief verloop	-

ID	UC8
Naam	Stel prioriteit voor vital parameter module in
Beschrijving	De gebruiker stelt de prioriteit in die bepaald welke abstracte numerieke module getoond wordt
Actoren	Gebruiker
Precondities	Een layout dat de vitale parameter module implementeerd is geopend in de wijzig layout pagina.
Verloop	1. Het systeem toont de layout in een grid 2. De gebruiker klikt op de vitale parameter module 3. Het systeem opent een formulier 4. De gebruiker breid in het opties veld het json object uit met de waarde ‘priority’ met een geheel getal 5. De gebruiker klikt op opslaan
Postcondities	-
Alternatief verloop	-

Bijlage D

Testrapport

1 Inleiding

In dit testrapport zal ik bespreken welke tests ik ga uitvoeren.

Ik focus vooral op het testen van de functionaliteiten. Dit zal ik doen door het uitvoeren van use case tests. Dit wordt verder besproken in hoofdstuk 2.

Het technische aspect van het systeem wordt getest door Unit tests. Deze test zijn geïntrigeerd in de cockpit en testen onderdelen van het system zonder afhankelijk te zijn van de rest van het systeem. Voor de unit tests wordt gebruik gemaakt van PHPunit. Dit framework maakt het schrijven van de test gemakkelijk. Wanneer alle tests gereeds zijn kunnen ze worden uitgevoerd. Hier worden ook de test die al eerder zijn gemaakt uitgevoerd. Alleen wanneer 100% van de tests slaagt is het onderdeel unittesten voltooid.

Als laatste testvorm maak ik gebruik van exploratory testing. Doordat het systeem door het visueel programmeren een oneindig aantal mogelijke input kan hebben is het onmogelijk op een 100% dekkingsgraad te behalen. De beste manier om het systeem te testen is exploratory testing. Hierbij gaat de tester bewust proberen het systeem kapot te maken door bijvoorbeeld merkwaardige invoer.

Use case tests

De use case tests heb ik opgesteld met behulp van het requirements document. Elke main flow en alternative flow van een use case wordt een test case. Ik maak hiervan eerst de logische testcase en dit breid ik vervolgens uit naar fysieke testcases.

Logische testcases

LOGISCHE CASE Bekijk tijdlijn module	**UC1**
actie	**Gewenste resultaat**
Gebruiker opent een dashboard dat de alert timeline module implementeert	Het systeem toont de tijdlijn module

LOGISCHE CASE Wijzig alarm	**UC2**
actie	**Gewenste resultaat**
Gebruiker opent een dashboard dat de alert timeline module implementeert	Het systeem toont de tijdlijn module
De gebruiker klikt op een alarm	Het systeem opent een formulier
De gebruiker vult het formulier in	-
De gebruiker klikt op opslaan	Het systeem slaat het alarm op in de database

LOGISCHE CASE annuleer wijziging	**UC2**
actie	**Gewenste resultaat**
Gebruiker opent een dashboard dat de alert timeline module implementeert	Het systeem toont de tijdlijn module
De gebruiker klikt op een alarm	Het systeem opent een formulier
De gebruiker vult het formulier in	-
De gebruiker klikt op annuleren	Het systeem slaat het alarm niet op in de database

LOGISCHE CASE acknowledge alarmering	**UC3**
actie	**Gewenste resultaat**
Gebruiker opent een dashboard dat de alert timeline module implementeert	Het systeem toont de tijdlijn module

Een alarm wordt actief en stopt daarna	Het systeem kleurt de tijdlijn geel en weergeeft de acknowledge knop
De gebruiker klikt op de acknowledge knop	Het systeem kleurt de tijdlijn terug naar de oorspronkelijke kleur en de acknowledge knop verdwijnt

LOGISCHE CASE Bekijk abstracte numerieke module	**UC4**
actie	**Gewenste resultaat**
Gebruiker opent een dashboard dat de abstracte numerieke module implementeert	Het systeem toont de abstracte numerieke module

LOGISCHE CASE Wijzig abstracte numerieke module	**UC5**
actie	**Gewenste resultaat**
De gebruiker opent de modulebeheerpagina	Het systeem toont de lijst met alle modules
De gebruiker klikt op de wijzig knop van een abstracte numerieke module	Het systeem opent een formulier
De gebruiker voert informatie in de velden in. De gebruiker klikt op de knop opslaan	De gewijzigde module is opgeslagen in de database

LOGISCHE CASE Wijzig abstracte numerieke module algoritme	**UC5**
actie	**Gewenste resultaat**
De gebruiker opent de modulebeheerpagina	Het systeem toont de lijst met alle modules
De gebruiker klikt op de wijzig knop van een abstracte numerieke module	Het systeem opent een formulier
De gebruiker klikt op de wijzig algoritme knop	Het systeem opent een venster waar een algoritme gebouwd kan worden
De gebruiker bouwt een algoritme en klikt op opslaan	Het venster wordt gesloten en het algoritme staat in het veld ingevuld

LOGISCHE CASE Bekijk vitale parameter module	**UC6**
actie	**Gewenste resultaat**
Gebruiker opent een dashboard dat de vitale	Het systeem toont de vitale parameter module

parameter module implementeert	

LOGISCHE CASE Stel prioriteit in voor de abstracte numerieke module	**UC7**
actie	**Gewenste resultaat**
De gebruiker opent de modulebeheerpagina	Het systeem toont de lijst met alle modules
De gebruiker klikt op de wijzig knop van een abstracte numerieke module	Het systeem opent een formulier
De gebruiker breid in het opties veld het json object uit met de waarde 'priority' met een geheel getal en klikt op opslaan	De prioriteit is opgeslagen bij de module in de database onder het opties veld.

LOGISCHE CASE Stel prioriteit voor vital parameter module in	**UC8**
actie	**Gewenste resultaat**
De gebruiker opent een layout dat de vitale parameter module implementeerd in de wijzig layout pagina.	Het systeem toont de layout in een grid
De gebruiker klikt op de vitale parameter module	Het systeem opent een formulier
De gebruiker breid in het opties veld het json object uit met de waarde 'priority' met een geheel getal en klikt op opslaan	De prioriteit is opgeslagen bij de layout_module in de database onder het opties veld.

Fysieke testgevallen

Ik heb geen tijd gehad om de logische testgevallen netjes om te zetten naar fysieke testgevallen. Omdat ik zelf de tests uitvoer is dit geen probleem. Wel is het in de toekomst wenselijk om echte fysieke testgevallen te hebben.