Eindverslag Bachelor Project
Modularisatie Monitor daemon

IN3405 BSc-Project
Faculteit Elektrotechniek, Wiskunde en Informatica van de TU Delft

Opdrachtgever: E.Novation, Capelle a/d IJssel

Auteurs:

David Hartveld	1181092
Piet-Jan Spaans	1217631

Commissie:

Bastiaan Bakker	Begeleider E.Novation
Peter Kluit	Begeleider TU
Kees Pronk	Coördinator bachelorprojecten TU

Datum: Vrijdag 13 augustus 2010

Voorwoord

Voor u ligt het verslag van ons bachelorproject voor de opleiding Technische Informatica aan de TU Delft. Wij hebben dit project zoals gebruikelijk ingevuld met een stageopdracht bij een ICT-bedrijf.

In ons geval was dit E.Novation, dat ons de mogelijkheid bood om hun dienstenmonitoring te verbeteren en gedeeltelijk te herschrijven naar een nieuw model. We zijn op 19 april begonnen, en hebben in totaal elf weken bij hen op de ontwikkelafdeling gewerkt.

Graag willen we hier iedereen bedanken die dit mogelijk heeft gemaakt, waaronder in het bijzonder de begeleider vanuit de TU, Peter Kluit en de begeleider vanuit E.Novation, Bastiaan Bakker.

David Hartveld en Piet-Jan Spaans

Inhoudsopgave

Voorwoord 2

Inhoudsopgave 3

1 Inleiding 4

2 De opdracht: Re-engineering van de monitor daemon 5

3 Het agile software ontwikkelproces van E.Novation 9

4 Overzicht nieuwe monitor daemon 17

5 Projectverloop 23

6 Evaluatie en aanbevelingen 26

7 Bibliografie 30

Appendix A Agile software ontwikkeling 31

Appendix B Technisch ontwerp 42

Appendix C Verklarende woordenlijst 46

Appendix D Oriëntatieverslag 47

Appendix E Plan van aanpak 65

1 Inleiding

Wij hebben ons bachelorproject uitgevoerd in de vorm van een stage bij E.Novation. E.Novation is een ICT-dienstverlener die actief is in het leveren van veilig berichtenverkeer voor de zorg, scheepvaart- en logistieksector. Hoewel de namen van de diensten verschillen, zijn veel diensten gebaseerd op de intern ontwikkelde E.Novation Message Service software of delen daarvan.

De E.Novation Message Service verwerkt voornamelijk gestructureerde berichten. Een voorbeeld uit de zorg is een digitale receptdoorgave van een huisarts aan een apotheker, maar ook complexere informatieuitwisseling tussen ziekenhuizen behoort tot de mogelijkheden. Een andere dienst, waar wij mee te maken hebben gekregen, is LSP Connect. Dit is een generieke 'stekker' oplossing waarmee derden hun eigen product kunnen aansluiten op het Landelijk Schakelpunt van het elektronisch patiëntendossier.

E.Novation bestaat uit vier business units: Managed Services, Solutions, Business Consultancy en International. De Managed Services unit is degene die de EMS levert, en waar ook Application Development onder valt, de afdeling waar wij stagelopen.

E.Novation heeft in Nederland twee kernteams binnen de afdeling Application Development: het maatwerkteam en het EMS team. De statistics collector en monitor daemon, waar wij mee te maken krijgen, vallen onder de verantwoordelijkheid van het EMS team. We hebben dan ook inhoudelijk het meest met hen te maken gehad. Naast de twee teams in Nederland heeft E.Novation ook nog een ontwikkelafdeling in China. Met de ontwikkelteams daar hebben wij niet te maken gehad.

De volgende hoofdstukken beschrijven verschillende aspecten van de stage. In hoofdstuk 2 beschrijven we de opdracht zoals die aan het begin van de stage was. In hoofdstuk 3 leggen we het Scrumproces bij E.Novation uit en hoe we dat in ons project toegepast hebben, in hoofdstuk 4 geven we een overzicht van de hiermee geleverde software. In hoofdstuk 5 een kort verslag van het verloop van de stage, met evaluatie en aanbevelingen in hoofdstuk 4. Daarna volgen nog appendices met onder andere een beschrijving van agile softwareontwikkeling met Scrum, het verslag van de oriëntatiefase en technische details van de geleverde software.

2 De opdracht: Re-engineering van de monitor daemon

E.Novation ontwikkelt de E.Novation Messaging Service (EMS), de basis voor een secure messaging service voor onder andere de zorgsector, onder de naam Zorgmail, en Rijkswaterstaat. De kwaliteit en beschikbaarheid van deze service worden bewaakt door een monitor- en een statistiekcomponent. Het doel van onze stage was deze componenten te herschrijven zodat ze onafhankelijk van de EMS zouden zijn, geïntegreerd, en minder belasting voor de bewakende server op zouden leveren. Centraal in de stage was het doel een (eind)product op te leveren dat geschikt zou zijn om in productie te nemen.

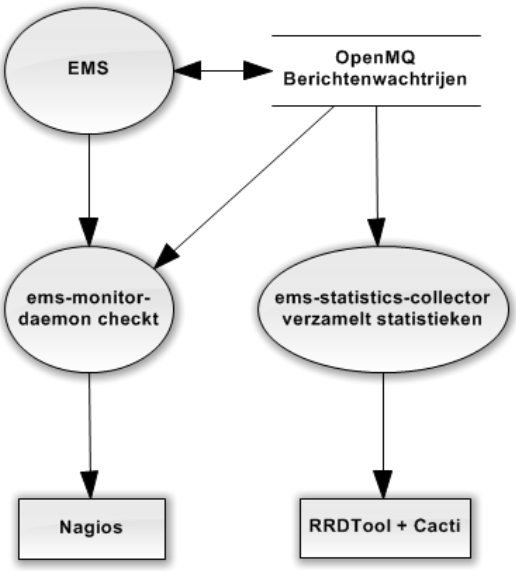
In dit hoofdstuk gaan we nader in op deze componenten zoals ze waren. We beschrijven aan welke veranderingen er behoefte was. Vervolgens werken we de doelen van de stage verder uit.

2.1 Het oorspronkelijke systeem

De EMS wordt op servicekwaliteit en beschikbaarheid gecontroleerd door een Nagios server. Deze server vraagt op reguliere intervallen de monitoring daemon om diensten binnen de EMS te controleren. Deze checkt daarop de beschikbaarheid van de gevraagde dienst en stuurt het verkregen resultaat terug naar de Nagios server.

De statistics collector in de EMS verzamelt gegevens over bijvoorbeeld de throughput van de service. Meestal gaat het dan om het aantal berichten per tijdseenheid dat door een proces in de EMS verwerkt wordt.

Beide componenten zijn essentieel bij het in gebruik hebben van de service en worden gebruikt om te controleren of de service goed werkt.

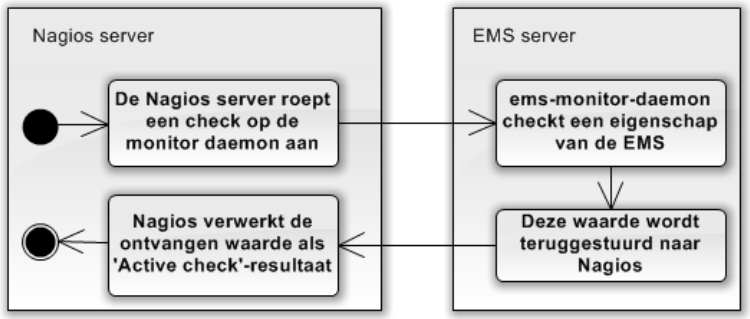


Figuur 1 Globaal dataflowdiagram

2.1.1 Monitor daemon

De monitor daemon wordt door Nagios benaderd om checks uit te voeren. Dit kan voor het testen van de beschikbaarheid van een proces zijn, zoals de meerderheid van normale Nagios checks. Het

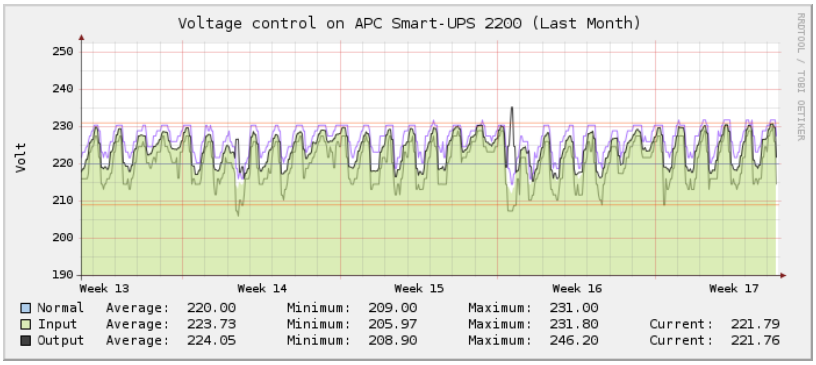
voordeel van een EMS-component die checks kan uitvoeren is dat er ook checks specifiek voor bepaalde processen gemaakt kunnen worden. Zo is er een check op de doorlooptijd van een bericht, van aankomst bij de EMS tot doorsturen, mogelijk. Doordat dit behandelen best even kan duren, kan het tot twee minuten kosten voor er een resultaat naar de Nagios server gestuurd kan worden. Het uitvoeren van een check door Nagios is zichtbaar gemaakt in het diagram in Figuur 2.



Figuur 2 Control Flow Diagram voor een Nagios check in het oorspronkelijke systeem

2.1.2 Statistics Collector

De statistics collector houdt zich bezig met het op regelmatige basis opslaan van bepaalde waarden, zoals het aantal threads in een EMS-proces of het aantal berichten in een queue. De component stuurt deze op naar een server waar ze in een round-robin-database komen. Van daaruit worden grafieken gegenereerd, die de data inzichtelijker maken dan alleen een check. E.Novation gebruikt hiervoor het programma RRDTool. Figuur 3 is een voorbeeld van een grafiek gegenereerd door RRDTool. Deze tool verzorgt het opslaan in round-robin-databases en het genereren van grafieken. Voor een grafisch overzicht van de grafieken wordt Cacti gebruikt, dat ook een handige grafische interface biedt voor het configureren van de vele commandline opties van RRDTool.



Figuur 3 Voorbeeld van een RRDTool grafiek

2.2 Problemen in het oorspronkelijke systeem

In het oorspronkelijke systeem speelden drie problemen: de code was sterk verweven met die van de EMS, de time-out van een check in Nagios was te snel en er waren capaciteitsproblemen op de Nagios server. Hieronder volgt een nauwkeurigere beschrijving.

In de oorspronkelijke situatie kon E.Novation de monitor daemon en statistics collector niet zonder de EMS draaien, doordat ze code uit de andere EMS componenten nodig hadden. Door het

loskoppelen van deze afhankelijkheden zouden de monitoring- en statistiekcomponenten ingezet kunnen worden bij andere systemen dan alleen de EMS.

Voor elke check op de eerder genoemde Nagios-server geldt een vaste time-out van anderhalve minuut. De mailloopcheck duurt vaak wel langer, tot vijf minuten, en kan daarom niet altijd goed draaien.

Bij het uitvoeren van de tests door Nagios wordt voor iedere test een nieuw proces gestart, dat een testresultaat teruggeeft. De schaalbaarheid van dit model is beperkt, omdat slechts een beperkt aantal processen tegelijkertijd kan worden uitgevoerd.

2.3 Integratie van statistiek en controles

De monitor daemon en statistics collector hebben overlap: beide componenten verzamelen nu gegevens over JMX-services. Voor deze services kan voortaan statistiek verzameld worden, waarbij er meteen na het verzamelen checks op uitgevoerd kunnen worden. Op deze manier kan de statistics collector in de monitor daemon geïntegreerd worden.

Verdere integratie is nog mogelijk door grafiekdata te genereren van de tijd die de huidige monitor daemon checks kosten en van het resultaat van die checks. Deze gegevens zijn namelijk al grotendeels beschikbaar in de monitor daemon, en kunnen na het samenvoegen ook opgestuurd worden naar RRDTool.

2.4 Doelen van de stage

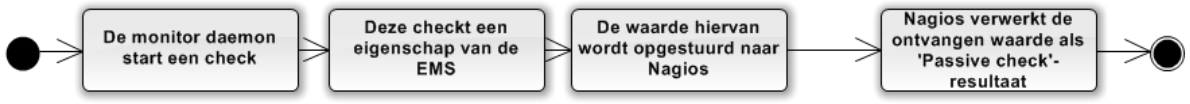
Aan het begin van de stage zijn de volgende doelen opgesteld:

– Het initialiseren van checks moet niet door Nagios, maar door de monitor daemon gebeuren

In het huidige systeem zijn er capaciteits- en time-out problemen op de Nagios-server, waar de checks gescheduled worden. Het voorstel is dit op te lossen door een omkering van de controle tussen Nagios en de monitoring daemon. Wij zullen Nagios vanaf de monitor daemon gaan voorzien van teststatus-events.

De tijdsbewaking ligt zo bij de monitor daemon, waardoor de time-outgrens configureerbaar wordt. De capaciteitsproblemen worden ook opgelost, doordat de checks nu decentraal opgestart worden, en Nagios alleen nog het verwerken hoeft te doen.

De controle in het nieuwe systeem is weergegeven in Figuur 4.



Figuur 4 Control Flow Diagram voor een Nagios check in het nieuwe systeem

– De statistics collector en de monitor daemon moeten geïntegreerd worden.

We zullen de monitor daemon en statistics collector integreren tot één applicatie, die en statistieken verzamelt, en statussen opstuurt naar Nagios. Hierdoor wordt niet alleen voorkomen dat er twee

keer data verzameld wordt, maar komt ook de mogelijkheid om statistieken over checks te verzamelen en checks op statistieken te doen.

– De nieuwe monitor daemon dient onafhankelijk van de EMS te worden.

De huidige componenten zijn nog afhankelijk. Door ze onafhankelijk te maken, kunnen ze breder ingezet worden. In de loop van de stage kregen we al de opdracht een eenvoudige versie te maken voor het product LSP Connect, iets wat anders niet mogelijk was geweest.

3 Het agile software ontwikkelproces van E.Novation

De afdeling Application Development bij E.Novation heeft verschillende ontwikkelteams. De teams werken volgens de Scrum methodiek, welke is ontwikkeld door Ken Schwaber en Kevin Sutherland (Schwaber & Sutherland, 2009). De Scrum methode moet gezien worden als een raamwerk dat door ontwikkelteams gebruikt kan worden om hun ontwikkelproces te structureren. Het geeft een kader waarin de planning van het ontwikkelproces wordt vastgelegd en beheerd. Het is niet een complete blauwdruk voor softwareontwikkeling, zodat ieder team zijn eigen invulling aan het raamwerk kan geven. Vaak worden best practices van de eXtreme Programming methodiek toegepast om dit gat in te vullen.

Dit hoofdstuk beschrijft de invulling van de Scrum methode zoals die bij E.Novation wordt toegepast, en zoals wij die hebben gebruikt tijdens onze stage. Een algemene inleiding tot Scrum kan worden gevonden in Appendix A. Daarnaast geven we, indien van toepassing, aan waar verschillen liggen tussen de invulling van Scrum die E.Novation gebruikt, en wat hierover in de literatuur is geschreven.

3.1 Rollen in het Scrum model

De literatuur beschrijft ontwikkelteams binnen Scrum als ‘cross-functional’ teams (Schwaber2010, p4): uit verschillende disciplines worden teamleden gezocht, die het product gaan uitontwikkelen. Bij E.Novation bestaat een ontwikkelteam uit vier of vijf team leden (Software Engineers), waaronder een Scrum Master. Het ontwikkelteam wordt bijgestaan door een een Product Owner. Verder is er een apart QA-team, dat ingezet wordt voor de uitvoering van acceptatietests, voordat de klant dit gaat doen.

De ontwikkelteams van E.Novation hebben niet allemaal dezelfde vorm. Ten tijde van onze stage bevatte het EMS team bijvoorbeeld geen functionele tester, terwijl het maatwerkteam er wel een had. De teams zijn deels “cross-functional”: een aantal van de rollen die in een ontwikkelteam nodig zijn, zoals software architect of programmeur, zijn aanwezig, maar andere ontbreken. Er wordt geprobeerd daar meer naar toe te groeien. Sinds de afloop van onze stage is het EMS team bijvoorbeeld uitgebreid met een functionele tester en een applicatiebeheerder.

Tijdens onze stage hebben we met twee personen een miniteam gevormd. Onze stagebegeleider, Bastiaan Bakker, heeft de rollen van Scrum Master en Product Owner op zich genomen. Ook heeft hij acceptance testing gedaan.

3.1.1 Ontwikkelteam

Tijdens onze stage hebben we als team van twee personen gefunctioneerd. Een van de belangrijkste aspecten van agile ontwikkelmethoden is de nadruk op interactie boven documentatie voor communicatie van ideeën tussen teamleden. Hoewel we met slechts twee personen plus onze stagebegeleider als Product Owner en Scrum Master een miniteam hebben gevormd, waren hiervan nu al de voordelen te herkennen: door onderling goed en direct te communiceren was het niet nodig om bijvoorbeeld complete functionele ontwerpen te schrijven, maar konden we ons beperken tot User Stories.

Een ander sterk aspect van Scrum is het concept van het cross-functional team. Omdat wij allebei dezelfde opleiding hebben gehad, hadden we grotendeels dezelfde kennis en vaardigheden. Maar

we konden ons niet specialiseren, omdat we van elkaar de opgeleverde producten (code, scripts, documentatie, etc.) moesten verifiëren. Als er meer teamleden waren geweest, was de impact van die tijdskosten kleiner geweest (door het aanleren van de benodigde vaardigheden te spreiden).

3.1.2 QA-team

Het QA-team is bij E.Novation een op zichzelf staand team, dat de beide ontwikkelteams ondersteunt met functionele acceptatietests. Testing wordt in de loop van de sprint gedaan per geïmplementeerde User Story. Aan het eind van de sprint wordt vervolgens het gehele opgeleverde product opnieuw bekeken of het aan de ontwikkelde tests voldoet.

Er is dus gekozen voor een combinatie van agile ideeen en meer traditionele ideeen. Er is geen sprake van een cross-functional team dat al het ontwikkelwerk doet, van ontwerp tot functioneel testen. Wel wordt een meer agile proces gevolgd door de manier waarop acceptance testing gedaan wordt. Het nieuw opgeleverde systeem wordt niet in een keer getest, maar wordt tijdens de Sprint per User Story getest. Hierdoor kan al tijdens de Sprint feedback worden geleverd door het QA team aan het ontwikkelteam.

Tijdens onze stage hebben we niet te maken gehad met het QA-team. Wel heeft Bastiaan Bakker deze functie uitgevoerd: hij heeft als Product Owner c.q. klant functioneel acceptatietesten gedaan. Bovendien heeft hij een technische acceptatietest gedaan: om te garanderen dat de kwaliteit van de opgeleverde code hoog genoeg was, heeft hij een groot deel van de opgeleverde code (inclusief unit tests) gecontroleerd. Overigens leverde dat op enig moment problemen op, doordat hij te weinig tijd had. Hierover staat meer in de evaluatie in paragraaf 6.1.3.

Aan het eind van onze stage hebben we bij een sprintdemonstratie van het maatwerkteam gezien dat E.Novation nog steeds bezig is met het verbeteren van het gebruikte ontwikkelproces. Er werd een presentatie gegeven door een teamlid dat een cursus voor Product Owners had gevolgd. Daarbij kwam naar voren dat het een goed idee is om QA te betrekken bij het opstellen van acceptatiecriteria van User Stories. Dit wordt nog niet expliciet op die manier gedaan: nu is het zo dat de User Stories ”over de muur worden gegooid”, en vervolgens gaat QA dan aan de slag. Door QA vanaf het begin te betrekken bij het opstellen van User Stories, wordt gedurende het complete ontwikkelproces gegarandeerd dat iedereen met dezelfde acceptatiecriteria werken, en dat de implementatie van User Stories vanaf het begin altijd de goede kant op gaat.

3.2 Management van ontwikkeling: Product Backlog en User Stories

Tijdens onze stage hebben we, zoals dat bij E.Novation gebruikelijk is, ons Product Backlog gevuld met User Stories. Deze zijn tot stand gekomen door met Bastiaan Bakker te overleggen over de richting die hij op wilde met het product. Vervolgens hebben wij User Stories opgesteld, en hebben we die aan hem voorgelegd. Daar kwamen dan vaak een aantal suggesties voor aanpassingen aan Stories, of extra Stories uit. Al snel hadden we een uitgebreid genoeg Backlog om de eerste Sprint te kunnen starten.

Aan het begin van iedere Sprint hielden we een Sprint Planning Meeting, waarbij we de planning voor die Sprint vastlegden. Zie hiervoor paragraaf 3.2.1. Aan het eind van de Sprint hielden we een Sprint Retrospective, waarin we besproken welke punten verbeterd konden worden ten opzichte

van de afgelopen Sprint. Ook hebben we twee keer een Sprint Review of Sprint Demo gehouden. Dit wordt respectievelijk in paragrafen 3.2.5 en 3.2.6 besproken.

3.2.1 Sprint Planning

Aan het begin van iedere Sprint hebben we een Sprint Planning gehouden. Tijdens die bijeenkomsten hebben we vastgelegd aan welke User Stories in die Sprint gewerkt zou gaan worden. Op basis van de geprioriteerde Product Backlog, waarvan per Story bekend was hoeveel tijd het zou kosten om die te implementeren, kon de Sprint Backlog worden vastgesteld.

Daarvoor moest eerst steeds de Product Backlog bijgewerkt worden. Tijdens de voorgaande Sprint ontdekten we vaak al Stories, maar tijdens de Sprint Planning probeerden we daar soms nog een aantal extra Stories bij te vinden. Vervolgens hebben we voor alle Stories met Planning Poker een aantal Story Points bepaald (zie A.3.1 voor meer informatie over Story Points en Planning Poker). Daarmee konden we schatten welke Stories we in de Sprint konden uitvoeren. De hoeveelheid tijd die we schatten voor de Stories werd bepaald door de definition of done: welke deliverables moeten we opleveren, en welke kwaliteit moeten ze hebben? In paragraaf 3.2.2 worden de Sprint deliverables besproken.

Op basis van onze inschattingen kon onze Product Owner, Bastiaan Bakker, de prioriteiten in de Product Backlog zonodig aanpassen. Hieruit volgde dan de Sprint Backlog voor de komende Sprint: De User Stories in de Product Backlog met de hoogste prioriteit, die op basis van onze inschattingen binnen de Sprint geïmplementeerd zouden kunnen worden.

3.2.2 Sprint deliverables

Om aan de “definition of done” te voldoen, zoals die bij E.Novation wordt gebruikt, moesten we onder andere source code met unit tests, configuratiebestanden, installatieinstructies, en eventuele extra documentatie in de vorm van wiki pagina’s opleveren. In deze paragraaf wordt dit in meer detail beschreven.

3.2.2.1 Implementatie van de User Story

Voor iedere User Story waarbij nieuwe code moet worden ontwikkeld, moest de betreffende source code, met bijbehorende unit tests en compile-time configuratiebestanden (builds op basis van Maven), naar de centrale source code management server (een Subversion repository server) worden gecommit. De source code wordt ontwikkeld met de Test-Driven Development (TDD) methode: nieuwe broncode wordt altijd met bijbehorende unit tests opgeleverd. Alleen dan wordt gecommit naar de centrale SCM server, als alle unit tests succesvol zijn. Dit is een harde eis omdat de source code altijd in releasebare vorm moet blijven. Immers, na iedere Story implementatie moet een releasebaar product beschikbaar zijn.

Tijdens onze stage hadden we te maken met een bestaande code, waar veelal de unit tests van ontbraken. We hebben steeds, bij aanpassing van die code, de ontbrekende tests allereerst ontwikkeld om vervolgens pas de daadwerkelijke aanpassingen te doen. Dit hebben we wederom gedaan om beter te kunnen garanderen dat het systeem zo min mogelijk fouten zou bevatten.

Om te controleren of de code, die geupload wordt naar de Subversion server, ook daadwerkelijk in een goede staat is, wordt een continuous integration server gebruikt (bij onze stage gaat het hierbij

om Hudson). Iedere nacht wordt een complete build uitgevoerd, inclusief het uitvoeren van de unit tests en een statische source code analyse met behulp van Sonar. Bovendien wordt, om daadwerkelijk snel feedback te krijgen op de staat van de source code tree, met regelmaat (bijvoorbeeld ieder uur) gecontroleerd of er nieuwe code gecommit is naar de SVN server, en dan wordt die code ook gebouwd. Als zo'n build vervolgens niet succesvol is, waarschuwt de Hudson server de ontwikkelaars per e-mail.

Deployment van (updates van) applicaties op servers vindt bij E.Novation plaats met behulp van RPM packages. Daarom horen de configuratiebestanden voor de RPM packaging tool ook bij de deliverables. Nieuw opgeleverde RPMs worden uiteraard ook uitgebreid getest: ze worden geïnstalleerd op de relevante servers in de ontwikkelomgeving. Die omgeving is vergelijkbaar met de productieomgeving, zodat alle opgeleverde software al tijdens de Sprint een soort praktijktest kan ondergaan.

3.2.2.2 Release notes per Story en per release

Voor iedere geïmplementeerde User Story wordt een installatieinstructie opgeleverd, waarin de story beschreven wordt. Daarbij hoort een algemene omschrijving van de Story, installatieinstructies, en een demonstratieinstructie van de nieuwe functionaliteit.

Voor een release worden alle instructies per story verzameld en in een definitieve Update Instruction samengevat. Hierin worden alle instructies opgenomen die nodig zijn om de release uit te rollen in de productieomgeving. Ook de demonstratieinstructies worden hierin opgenomen, zodat na het uitrollen getest kan worden of de nieuwe release inderdaad op de juiste wijze opgebracht is.

3.2.2.3 Documentatie op de interne wiki

Alle relevante documentatie wordt op de interne wiki geplaatst. Het gaat primair om de hierboven genoemde release notes. Alleen in bijzondere gevallen worden andere dingen gedocumenteerd.

Tijdens onze stage hebben we wat uitgebreider verslag gedaan van onze werkzaamheden. Dit was belangrijk, omdat anders de door ons verzamelde kennis direct weer verloren zou gaan. Daarom hebben we een aantal pagina's met relevante informatie op de wiki geplaatst over de werking van de nieuwe daemon 'enomon', met primair aandacht voor nieuw geïntroduceerde technologie, zoals het NSCA-protocol.

3.2.3 Monitoring van voortgang op de Story Wall

Tijdens de Sprint wordt voortgang ten opzichte van de Sprint Backlog bijgehouden op een muur, de Story Wall. De Stories worden geprint op briefjes van 11x8 cm. Ze worden in een aantal kolommen opgehangen, die representeren in welke fase de ontwikkeling van die specifieke Story zich bevindt. Al naar gelang een Story verder door het ontwikkeltraject beweegt, zal het printje ook van kolom naar kolom verplaatsen.

De TODO kolom bevat alle Stories die nog geïmplementeerd moeten worden: de Sprint Backlog. Aan het begin van de Sprint bevat deze kolom alle Stories, in de volgorde van prioriteit uit de Sprint Backlog. Als een Story vervolgens geïmplementeerd wordt, verplaatst die naar de kolom DOING. Omdat een ontwikkelaar nooit aan meer dan één Story tegelijkertijd kan werken, bevat de kolom DOING nooit meer Stories dan er ontwikkelaars zijn. Op enig moment bepaalt de ontwikkelaar die de

Story implementeert, dat hij klaar is. De Story wordt dan verplaatst naar de kolom TO VERIFY: een andere ontwikkelaar moet verifiëren of de implementatie van de Story voldoet aan de “defintion of done”. Als dit dan gebeurd is, wordt de Story naar de kolom QA verplaatst: er moet acceptance testing worden gedaan door het QA-team. Nadat de implementatie van de Story geaccepteerd is, kan deze naar de laatste kolom, DONE, verplaatst worden, en is de Story gereed om in een release te worden opgenomen.

Er zijn verschillende momenten waarop een Story teruggeplaatst kan worden naar een eerdere fase (kolom). Ten eerste kan het voorkomen dat tijdens de implementatiefase (DOING) een obstakel wordt gevonden, waardoor de Story op dat moment niet direct verder kan worden ontwikkeld. De Story wordt dan afhankelijk van de ernst van de blokkade naar de zijkant van de kolom verplaatst, of naar de TODO fase teruggeplaatst, als er niet op zeer korte termijn een oplossing kan worden gevonden. Ook kan een Story, die niet door de VERIFICATIE fase of de QA fase heen komt, teruggeplaatst worden naar DOING of zelfs TODO.

Tijdens onze stage hebben we de User Stories die door een van ons geïmplementeerd werden, door de ander later verifiëren. Bastiaan Bakker heeft acceptance testing gedaan. We zijn enkele keren een geval van negatieve acceptatie tegengekomen. Een mooi voorbeeld hiervan was de implementatie van een workaround voor een bug in OpenMQ. In eerste instantie dachten we dat we een werkende workaround hadden geïmplementeerd, en daarom hadden we (na testing door de implementator) de Story al naar de verificatiefase verplaatst. Maar tijdens de verificatiefase bleek dat de fout zich bleef manifesteren, en dus werd de story terugverplaatst naar de Sprint Backlog, om vervolgens als eerste weer opgepikt te worden om te worden uitgevoerd.

3.2.4 Automatisering van Backlog en Story Wall: XPlanner

Tijdens onze stage hebben we de tool XPlanner gebruikt om onze Backlogs te managen. XPlanner maakt het mogelijk om een Sprint Backlog vast te stellen, uit te breiden, of anderszins aan te passen. Vervolgens kan een Sprint gestart worden, waarna per User Story, per taak binnen die Story tijd kan worden geschreven en aangegeven kan worden hoeveel tijd er nog nodig is om een taak, en daarmee een Story, te implementeren.

Met deze informatie kan de voortgang van de Sprint goed gecontroleerd worden: op ieder moment is inzichtelijk hoeveel tijd nog nodig is om de verschillende Stories, die in de Sprint Backlog staan, en die op dat moment geïmplementeerd worden, daadwerkelijk tot een releasebare staat te brengen. Daarbij hoort een Sprint release burn down, een grafiek waaraan in een oogopslag te zien is, hoe ver het team nog verwijderd is van een complete release. Bovendien is hierdoor duidelijk te zien of de snelheid van implementatie van Stories hoog genoeg is. Dit wordt ook wel de Sprint Velocity genoemd.

Met XPlanner hadden we duidelijk inzicht in de hoeveelheid tijd die nog nodig was om de Sprint Backlog te implementeren. Dat hielp goed bij het bepalen of bepaalde Stories wel of niet binnen de Sprint geïmplementeerd zouden gaan worden. Helaas was er een probleem met de XPlanner software waardoor voor onze Sprints geen Sprint Burn Down Chart kon worden weergegeven. Daardoor was lastig te bepalen wat daadwerkelijk onze Sprint Velocity was. Dat is jammer, want dit

is eigenlijk een leidend metriek binnen het Scrum raamwerk om te bepalen of het ontwikkeltempo nog wel hoog genoeg is.

3.2.5 Sprint retrospective

Onze Sprint Retrospectives duurden normaliter ongeveer een half uur. Daarbij hebben we een eenvoudige vorm aangehouden, in navolging van E.Novation.

Ieder teamlid, inclusief de Scrum Master en Product Owner, beschrijft kort drie positieve en negatieve aspecten van de afgelopen Sprint. Vervolgens kiest ieder teamlid welke punten hij het belangrijkst vindt om te bespreken: ieder teamlid mag drie gewichtjes verdelen over de verschillende positieve en negatieve punten. Voor zover er dan tijd is, worden de punten, op aantal gewichtjes, besproken.

Op deze manier worden de belangrijkste pijnpunten én positieve punten gevonden. Voor de problemen moeten oplossingen worden bedacht, die dan in de volgende Sprint kunnen worden toegepast. De positieve punten worden ook besproken, zodat die indien nodig nog verder versterkt kunnen worden. Zo komt er uit de Sprint Retrospective een aantal verbeterpunten, dat de volgende Sprint weer beter uitvoerbaar kan maken. Door iedere Sprint een Retrospective te houden, wordt het ontwikkelproces steeds weer iets verder verbeterd.

3.2.6 Sprint Demo

Bij iedere Sprint, waarin een release wordt opgeleverd, hoort ook een Sprint Demo. Wij hebben zelf twee demo's gehouden, na de derde sprint, na zes weken, en na Sprint 5, aan het eind van onze stage. We hebben na de eerste twee Sprints geen demonstratie gehouden omdat we toen nog geen concreet releasebaar product hadden. In de vierde sprint hebben we het releasedoel niet gehaald en dus hebben we na de vijfde sprint het eindresultaat gepresenteerd.

De Sprint Demo wordt bij E.Novation gebruikt voor meerdere doelen. Ten eerste wordt getoond aan de afdeling wat het ontwikkelteam concreet bereikt heeft in de afgelopen Sprint. Welke doelen zijn behaald in termen van gecreëerde toegevoegde waarde? Wat heeft het ontwikkelteam aan de klant geleverd? Hierbij wordt meestal een (interactieve) presentatie gegeven van de ontwikkelde software. Verder wordt vaak besproken wat de belangrijkste, leerzaamste of interessantste ervaringen van het team waren in de Sprint. Daarnaast wordt aangegeven wat nog niet geïmplementeerd is, en wat er op korte termijn ontwikkeld gaat worden. Tot slot wordt soms nog besproken hoe extra toegevoegde waarde gecreëerd zou kunnen worden, die vermarkt kan worden.

3.3 Evaluatie toepassing Scrum tijdens de stage

We hebben een aantal positieve en negatieve punten geïdentificeerd aan het ontwikkelproces zoals we dat tijdens onze stage hebben toegepast. In deze paragraaf bespreken we alle Scrumgerelateerde zaken. In Hoofdstuk 4 evalueren we de stage in het algemeen.

3.3.1 Voordelen

Een van de belangrijkste redenen om voor een agile ontwikkelproces te kiezen, is dat het eenvoudiger is om gaandeweg aan te passen aan veranderende wensen van de klant. Tijdens onze stage was E.Novation zelf klant. In grote lijnen stond in onze opdracht al vast wat we gingen

ontwikkelen. Maar op gedetailleerder niveau waren er nog veel onduidelijkheden. Dit manifesteerde zich in de loop van de stage in de vorm van veranderende prioriteiten, van Sprint tot Sprint. Het mooie van het Scrum raamwerk is dat hier prima mee kan worden omgegaan: al naar gelang de prioriteiten anders zijn, wordt de Product Backlog aangepast. Dit is een sterk punt van Scrum, dat zich duidelijk in de praktijk toont.

Een ander sterk punt is het gebruik van de Story Wall. Hierop is op ieder moment heel duidelijk te zien wat de stand van zaken is. Wat is de planning voor de Sprint? Waar zijn we op dit moment mee bezig? Wat moet door wie (ontwikkelteam, QA) worden gedaan? Wat is al klaar? Het zorgt er voor dat alle betrokkenen weten hoe het met de Sprint staat.

Het derde grote positieve punt van Scrum sluit daarbij aan. De Daily Standup Meeting zorgt ervoor dat het hele team iedere dag even op de hoogte gebracht wordt van wat er gedaan is, gedaan gaat worden, en waar de problemen op dat moment liggen. Zo is iedereen op de hoogte van wat er op dat moment speelt en worden de communicatielijnen kort gehouden, ook naar de Product Owner toe, die anders toch wat verder van het team afstaat. Dit is zeer waardevol geweest, omdat we tijdens DSMs wel eens tot de conclusie kwamen dat er bijgestuurd moest worden.

Ten slotte is het Scrum raamwerk heel flexibel ten aanzien van veranderende omstandigheden binnen het ontwikkelteam of de organisatie waarin het team opereert. Sprints duren maar een korte tijd, en na iedere Sprint wordt een Sprint Retrospective gehouden waarin de Sprint wordt geëvalueerd. Dit biedt gelegenheid tot aanpassing van het proces aan de omstandigheden, waardoor de volgende Sprint weer efficiënter kan worden gewerkt. Dit is belangrijk om de Sprint Velocity op peil te houden.

3.3.2 Nadelen

Het was een nadeel dat we slechts met twee personen een team vormden. Omdat de definition of done stelde dat een ander teamlid altijd moet controleren wat er geproduceerd wordt, betekende dit dat er geen ruimte was voor specialisatie. Hierdoor moesten we allebei complete kennis hebben. Dit was heel kostbaar, en niet voor niets wordt in literatuur aanbevolen om met teams van 5 tot 9 personen te werken.

Een ander nadeel van de Scrum aanpak met behulp van User Stories is dat de focus ligt op het opleveren van incrementele verbeteringen aan het product, en zodoende ook aan de documentatie. Maar bij E.Novation blijft documentatie over het algemeen beperkt tot het uitwerken van Release Notes, tenzij de klant hier expliciet eisen aan stelt. Overkoepelende documentatie van bestaande systemen bestaat in beperkte vorm op een interne wiki, en vooral ook als ontastbare kennis bij de betreffende ontwikkelteams. Als de klant wel eisen aan documentatie stelt, worden die uiteraard ingevuld. Een voorbeeld is documentatie op basis van de JSTD-016, een internationale standaard voor specificatie, ontwerp- en beheerdocumentatie.

Aan het eind van onze stage hebben we gericht een aantal onderwerpen op de interne wiki gedocumenteerd, zodat de door ons opgebouwde kennis overgedragen kon worden. Bij agile ontwikkelmethoden wordt communicatie en interactie tussen teamleden verkozen boven het aanleggen van uitgebreide documentatie. Maar documentatie bij (kortlopende) projecten waarbij

een tijdelijk team wordt opgetuigd, en de teamleden daarna weer uit de organisatie verdwijnen, is onontbeerlijk als de organisatie die met de software verder moet, niet opnieuw wil beginnen. We hebben op de laatste dag van onze stage dan ook een overdrachtsmoment gehouden met twee leden van het EMS team, om ook op die manier kennis over te dragen.

Een derde nadeel van onze aanpak was dat er – buiten de Product Backlog – geen goed overzicht was van de concrete, inhoudelijke doelen van het (stage)project. Het hebben van een Story Map of een vergelijkbaar overzicht – bij voorkeur zo groot mogelijk aan de muur – zou erg geholpen hebben bij het overzicht houden over het complete project (zie A.3.1.3 voor een toelichting op Story Mapping). Helaas werden we pas in de laatste weken van onze stage bekend met deze methode. Maar bij een volgend project zouden we zeker zo'n methode gebruiken.

Tot slot is er een risico aan te wijzen bij het gebruik van User Stories. De Scrum aanpak werkt, mits aan de voorwaarden voor het opstellen van User Stories wordt voldaan. Maar als dat niet gebeurt, loopt het team risico's ten aanzien van de inschattingen die ze doen. Een onduidelijke of te grote Story is moeilijk in te schatten. Tijdens Sprint 4 hebben we hiermee problemen ondervonden. We hebben toen een veel te grote Story gedefinieerd, en die niet opgesplitst in kleinere Stories. Hierdoor hebben we vele uren extra besteed aan het implementeren van de Story, waardoor we de uiteindelijke release van die Sprint niet gehaald hebben. Het is dus van essentieel belang om User Stories zo nauwkeurig en klein mogelijk te definiëren. Dat maakt het inschatten makkelijker én nauwkeuriger.

3.3.3 Conclusie

Over het algemeen zijn we heel enthousiast over het toepassen van het Scrum raamwerk in onze stage. We hebben erg moeten wennen aan de functionele en resultaatgerichte wijze van formuleren van User Stories – in het begin hebben we vaak de fout gemaakt om in technische termen Stories te formuleren. Gelukkig was onze stagebegeleider Bastiaan als Product Owner er dan om ons daarop te wijzen. Toen we eenmaal de slag te pakken hadden, werd ons steeds meer duidelijk dat de flexibiliteit die het Scrum raamwerk biedt, bij dit soort projecten erg nuttig kan zijn. We denken dat we veel geleerd hebben van deze aanpak en een stukje kennis hebben opgedaan over software ontwikkelingsprocessen, die op de universiteit nog niet zo sterk aan de orde is geweest.

4 Overzicht nieuwe monitor daemon

Tijdens onze stage hebben we op basis van de bestaande EMS monitor daemon en statistics collector een nieuw zelfstandig project opgeleverd: enomon, de nieuwe E.Novation Monitor Daemon. Het opgeleverde project bestaat uit een kernel, en twee modules voor LSP Connect en Zorgmail. De modules, en daarmee ook de kernel, zijn gebouwd voor de productieomgeving van beide producten.

De nieuwe daemon heeft twee hoofdtaken: het verzamelen van data over de EMS, en het verwerken en doorsturen van de verzamelde data. De statistics collection is nu geïntegreerd met het monitoren, en het resultaat van een uitgevoerde check wordt door de monitor daemon naar Nagios opgestuurd.

De architectuur van de applicatie zoals beschreven in hoofdstuk 2 is opgezet. Er worden zelfstandig updates naar Nagios gestuurd, enomon is onafhankelijk van de EMS, en de monitor daemon en statistics collector zijn in zoverre geïntegreerd dat er checks op statistiekdata plaatsvinden. Het doel om de volledige monitor daemon functionaliteit te bieden is niet gehaald, evenals het doel om statistiek over die checks te verzamelen. In dit hoofdstuk geven we een overzicht van het ontwerp en de werking van de verschillende onderdelen. Voor een gedetailleerde technische beschrijving van de monitor daemon, zie Appendix B.

4.1 Architectuur

De architectuur van de nieuwe monitor daemon bouwt voort op die van de statistics collector, waar al veel van de benodigde functionaliteit aanwezig was: JMX-data verzameling, het publiceren daarvan in een Queue, en het opsturen van de verzamelde data naar RRDTool.

Om bewerkte data opnieuw te laten verwerken, en bewerkingen op meerdere data-objecten tegelijk te doen, was het nodig om deze structuur uit te bouwen. Hierbij is ervoor gekozen het publish-subscribe pattern te behouden en uit te breiden. Het publish-subscribe pattern is hier handig, omdat tijd, ruimte en synchronisatie loskoppelen (Eugster, 2003) alledrie voordelen opleveren.

Hierbij is het loskoppelen van tijd dat de publisher en subscribers asynchroon kunnen werken, berichten hoeven niet meteen gelezen te worden zodra ze verstuurd zijn. Het loskoppelen van synchronisatie wil zeggen dat het verzenden en ontvangen van berichten de control flow van de applicatie niet meer hoeft te onderbreken, doordat dit in andere threads kan gebeuren. Het loskoppelen van ruimte houdt in dat publishers en subscribers elkaar niet meer hoeven te kennen.

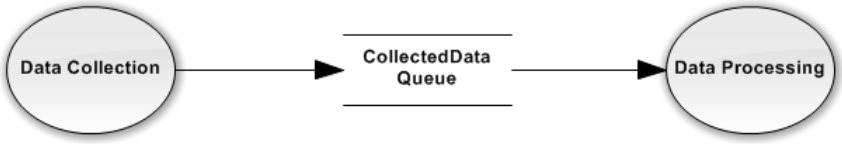
Door het loskoppelen van tijd en synchronisatie hoeft het versturen van updates niet te wachten op het ophalen van de JMX-data. Door een nieuw punt te introduceren om verwerkers aan te koppelen wordt ook de ruimte losgekoppeld: het wordt makkelijker om meerdere publishers en subscribers te hebben. Ook wordt het hiermee mogelijk dat een klasse meerdere data-objecten verwerkt, iets wat nodig is voor een check op meerdere services.

Het verwerken van meerdere data-objecten is niet mogelijk zonder een vorm van data filtering. Bij het publish-subscribe pattern is er de keuze om berichten te filteren op topics of op content.

Er zou in de nieuwe monitor daemon voor elke check een topic zijn, en een topic met alle data. Door geen topics te beheren, maar een algemene queue met content filtering te gebruiken, wordt onnodig werk voorkomen. De content wordt gefilterd aan de hand van de attributen van de data-objecten.

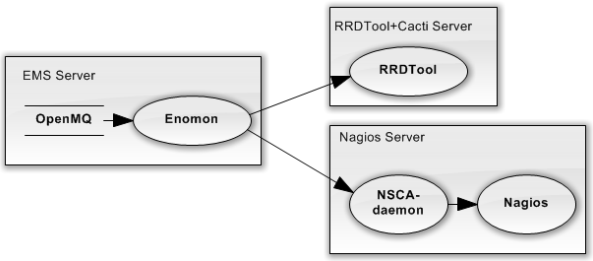
De queueconstructie is dus uitgebreid tot een volledig publish-subscribe pattern. De data die in de queue gepubliceerd wordt, heeft een naam-attribuut waar de subscribers de data op kunnen filteren. Hoewel volgens het pattern niet de taak van de subscribers, was het hier een eenvoudige manier om sommige subscribers (voor RRD) alle data te laten krijgen, en anderen alleen specifieke data (voor Nagios checks). Uiteindelijk bleek filtering ook voor RRD-updates nodig, en zou het filteren wel beter op zijn plek zijn in een beheerklasse tussen de queue en de subscribers.

De nieuwe monitor daemon bestaat daarmee uit twee belangrijke elementen, data-verzameling en data-verwerking, gescheiden door een publish-subscribe structuur. Die structuur is als queue te zien in Figuur 5.



Figuur 5 Globaal Data Flow Diagram voor de nieuwe monitor daemon

Vergeleken met de statistics collector, bestaat de data-verwerking eruit dat de data niet alleen naar de RRDTool server wordt gestuurd, maar ook wordt vergeleken met een drempelwaarde, waarvan het resultaat naar de Nagios server gestuurd wordt. Zie Figuur 6 voor een overzicht van de servers en programma's die gebruikt worden.



Figuur 6 Server overzicht bij de nieuwe monitor daemon

We gaan in de volgende paragrafen in op de verschillende subsystemen van enomon: data-verzameling, data-verwerking, het opsturen van een checkresultaat naar Nagios, configuratie en de opgeleverde enomon-modules.

4.2 Dataverzameling

De monitor daemon verzamelt de gegevens van EMS-processen en OpenMQ-queues via JMX, net zoals de oude statistics collector, maar is uitgebreid met de mogelijkheid de leeftijd van het eerstvolgende bericht in een gegeven OpenMQ-queue uit te rekenen en te verzamelen. De

verzamelde gegevens worden opgeslagen in een CollectedData-object dat in de queue wordt gezet, zoals te zien in Figuur 7 en Figuur 8.

Figuur 7 Data Flow Diagram van dataverzameling

Figuur 8 Klassendiagram van dataverzameling

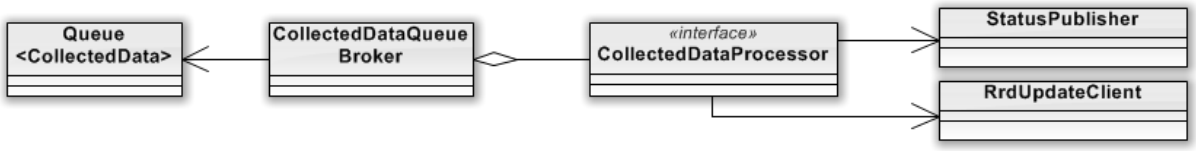
De data wordt verzameld door klassen die de interface Statistics Collector implementeren. Elke Statistics Collector wordt aangeroepen door een CollectedDataPublisher, de klasse die het geproduceerde dataobject op de interne data-queue zet.

Om een probleem met gelijktijdige verbindingen in OpenMQ te voorkomen, worden alle CollectedDataPublishers in één thread achter elkaar uitgevoerd. Dit is minder efficiënt dan ze parallel uitvoeren, maar voorkomt de problemen. De hoogst haalbare efficiëntie van het verzamelen zou bereikt kunnen worden door de JMX-verbinding te delen tussen de CollectedDataPublishers, die nu iedere keer een nieuwe verbinding maken.

De thread wordt op zijn beurt door het scheduling framework Quartz gestart. Dit gebeurt met een regulier interval. Dit interval kan aangepast worden, afhankelijk van de tijd die het kost om alle opdrachten uit te voeren.

4.3 Data-verwerking

De gegevens zijn verzameld, en staan in de queue. De processors verwerken de gegevens uit de queue. Enerzijds wordt de verzamelde data naar de RRDTool-grafiekserver opgestuurd. Anderzijds wordt de data vergeleken met drempelwaardes om de daaruitvolgende status op te sturen naar de Nagios server.



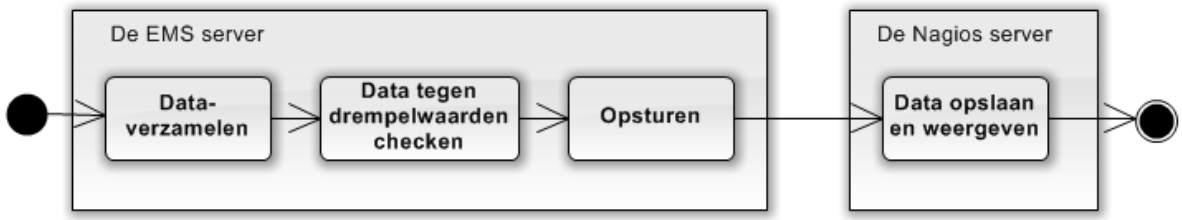
Figuur 9 Klassendiagram van dataverwerking

De data wordt door de Broker aan alle geregistreerde CollectedDataProcessors gegeven. Implementaties van deze interface CollectedDataProcessor zijn er voor het versturen van updates naar de RRDTool-server, en voor het checken tegen drempelwaarden voor Nagios. Het checken tegen drempelwaarden gebeurt in de klasse SimpleThresholdCheck. Deze drempelwaarden zijn per check configureerbaar. De binnenkomende data wordt gefilterd op naam, zodat de juiste drempelwaarden op de juiste data worden toegepast.

Om de leeftijd van een bericht netjes in uren, minuten en seconden weer te geven, heeft de SimpleThresholdCheck een formatter. Bij de initialisatie wordt bepaald of de standaard formatter of de leeftijdsformatter nodig is.

4.4 Status naar Nagios opsturen

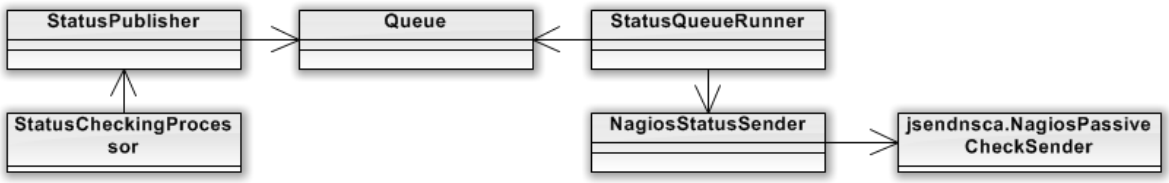
Het resultaat van het checken van de gegevens in een dataobject is een Status voor Nagios. Deze wordt op het EMS-systeem gemaakt, en moet op de Nagios server verwerkt worden, zie Figuur 10.



Figuur 10 Verdeling van verwerkingsstappen over de twee servers

Er was niet bepaald hoe de Status precies opgestuurd zou worden. Er kwamen uit ons onderzoek daarnaar twee mogelijke protocollen: NSCA en SNMP. Het Nagios Service Check Acceptor-protocol is een eigen protocol van Nagios om zelf Statussen naar Nagios op te kunnen sturen. Een NSCA-daemon voor de server wordt op de Nagios site aangeboden. Het Simple Network Management Protocol, dat door meer netwerkbeheertools ondersteunt wordt, biedt ook de mogelijkheid om 'passieve' updates te sturen van de status van een service. Deze mogelijkheid wordt echter niet door Nagios zelf ondersteund, maar moet eerst vertaald worden naar een Nagios-formaat.

We hebben voor het NSCA-protocol gekozen omdat dit beter door Nagios ondersteund wordt, en afhankelijkheid van Nagios geen bezwaar was. Om NSCA-updates te kunnen sturen, hebben we de Open Source JSendNSCA-bibliotheek gebruikt. Voor het ontvangen hebben we de door Nagios geleverde NSCA-daemon geinstalleerd op de Nagios-server.



Figuur 11 Klassendiagram van het status opsturen

De te versturen Status wordt in een Status-Queue geplaatst. Van daaruit wordt de data verstuurd als er meer dan een bepaald aantal objecten beschikbaar zijn, of als er een bepaalde tijd verstreken is.

De status updates worden dus niet een voor een verstuurd, maar in bulk. Dit lost performance problemen op: voor iedere status update moet een nieuwe netwerkverbinding worden opgebouwd. Het verwerken van de status updates duurt dan tot enkele honderden milliseconden. Nu kan deze vertraging beperkt worden tot een keer per bulk verzending.

Het aantal objecten in de bulk staat nu vast op 50. Dit zou configureerbaar gemaakt kunnen worden, zodat erop bij te sturen is afhankelijk van de omgeving waarin de applicatie en Nagios opgesteld worden.

De JSendNSCA bibliotheek ondersteunt het versturen van status updates in bulk (nog) niet. Door middel van een patch hebben wij deze functionaliteit eraan toegevoegd. Dit was een mooi voorbeeld van een van de voordelen van Open Source software, zoals die door E.Novation gezien worden.

4.5 Configuratie

De unieke naam van het CollectedData-object moet en bij dataverzameling en bij dataverwerking ingesteld worden om de interne communicatie goed te laten verlopen. Vanwege de hoeveelheid aan verzamelingen en checks die er, liefst zo makkelijk mogelijk, gemaakt moet kunnen worden, was een makkelijke configuratie nodig.

De combinatie van deze twee factoren heeft geleid tot een configuratie waarbij alle checks ingesteld worden afhankelijk van of en hoe ze met een paar regels in een configuratiebestand gedefinieerd staan.

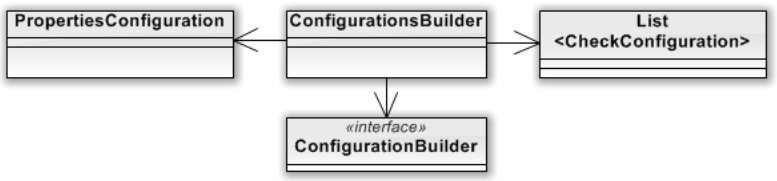
Er zijn drie soorten checks te configureren: checks op willekeurige JMX-gegevens, checks op gegevens van berichtenqueues, en checks op de leeftijden van het oudste bericht in berichtenqueues.

De te verzamelen data, en de checks daarop, zijn eenvoudig te configureren door middel van een configuratiebestand. In een configuratiebestand wordt het verzamelen en checken met een paar regels gedefinieerd. Zie de oude Spring XML-configuratie en de nieuwe configuratie in Table 1.

<bean id="openmqJmxCollector" abstract="true" class="nl.enovation.commons.statistics.collector.AbstractJmxCollector"> <property name="serviceUrl" value="${openmq.jmx.serviceUrl}" /> <property name="connectionFactory" ref="failoverJmxConnectionFactory" /> </bean> <bean id="openmqJmsQueue" abstract="true" class="nl.enovation.commons.statistics.collector.JmxCollector" parent="openmqJmxCollector"> <property name="queue" ref="queue" /> <property name="attributeNames"> <list> <value>NumMsgs</value> <value>NumMsgsIn</value> <value>NumMsgsOut</value> </list> </property> </bean> <bean id="openmqArchiveQueueCollector" parent="openmqJmsQueue"> <property name="name"> <value>activemq-queue-ems-archive</value> </property> <property name="mbeanName"> <value> com.sun.messaging.jms.server:type=Destination,subtype=Monitor,desttype=q, name="EMS.ARCHIVE" </value> </property> </bean>	emsJmsCheck.emsArchive.serviceUrl=${openmq.jmx.serviceUrl} emsJmsCheck.emsArchive.queueName=EMS.ARCHIVE emsJmsCheck.emsArchive.rrdName=activemq-queue-ems-archive emsJmsCheck.emsArchive.NumMsgs.serviceName=EMS Q Archive Size emsJmsCheck.emsArchive.NumMsgs.warning=250 emsJmsCheck.emsArchive.NumMsgs.critical=1000 emsJmsCheck.emsArchive.NumMsgsIn= emsJmsCheck.emsArchive.NumMsgsOut=

Table 1 Configuratie van één check: links in Spring XML, rechts in de nieuwe configuratie

De centrale configuratie met CheckConfiguration en Manager zorgt ervoor dat instanties van de verzamelende en verwerkende klassen op de goede plek en met de juiste instellingen opgestart worden.



Figuur 12 Klassendiagram van de configuratie

Uit het genoemde propertiesbestand wordt een lijst met CheckConfigurations gecreëerd. De zo geproduceerde lijst van CheckConfiguration instanties wordt vervolgens door Manager en Factory-objecten gebruikt om de uitvoerende objecten te initialiseren en aan elkaar te hangen.

De hele applicatie werkt, zoals de componenten waarop voortgebouwd is, met het Spring framework. Alle instanties van klassen zijn in XML-bestanden gedefinieerd, en worden door Spring opgestart. Spring verzorgt hierbij Dependency Injection, zodat niet handmatig objecten hoeven te worden geïnitialiseerd in constructors en initialisatiemethoden.

Omdat vooraf onbekend is hoeveel checks er nodig zijn, wordt deze dependency injection hier gedeeltelijk vervangen door zelf geschreven initialisatie in de Manager- en Factory-klassen.

De JMX en de leeftijdchecks worden gedeeltelijk apart geïnitialiseerd (eigen CheckConfiguration, eigen Managers), omdat ze op verschillende manieren data ophalen en weergeven. Het verwerken is op het weergave-format na gelijk.

4.6 Modules

De monitor daemon bestaat uit een kernel-module en modules met voorgeconfigureerde checks voor specifieke systemen. Tijdens de stage hebben wij modules voor LSP Connect en Zorgmail opgeleverd.

Bij het opstarten laadt de kernel alle aanwezige modules in. Het dynamisch inladen van modules is overwogen, maar niet geïmplementeerd. Een uitgebreide overweging van deze keuze is te lezen in het Oriëntatieverslag in Appendix D. Hiervoor was de keuze uit OSGi en twee kleinere frameworks. De voorkeur ging wat betreft kwaliteit, ondersteuning en flexibiliteit uit naar OSGi. Uiteindelijk bleek dit, bij het inschatten van de ontwikkeltijd, teveel werk ten opzichte van de toegevoegde waarde voor E.Novation in deze applicatie.

Een module bestaat uit properties-bestanden met de nieuw geconfigureerde checks en xml-bestanden om de properties-bestanden in te lezen en de checks te initialiseren. Bij het installeren van een module door middel van een RPM worden deze automatisch op de goede plek gezet, zodat ze bij een herstart van de monitor daemon service ingeladen worden.

5 Projectverloop

In dit hoofdstuk staat een verslag van het verloop van ons project, opgedeeld in Sprints. We noemen de belangrijkste zaken die tot blokkades en daarmee vertraging hebben geleid.

5.1 Sprint 1 – Oriëntatie

In de oriëntatiefase hebben we het oriëntatieverslag in Appendix D geschreven. In deze Sprint hebben we ons gericht op het onderzoeken van de technologische mogelijkheden om de stagedoelen te bereiken. Hierbij is gekeken naar het systeem zoals dat in gebruik was, de toekomstige manier van communiceren tussen de monitor daemon en Nagios, en hoe het OSGi-platform het dynamisch inladen van nieuwe modules zou kunnen faciliteren.

Door het vereiste onderzoekende karakter van deze week, hebben we ons minder bekend gemaakt met de technologie dan in de tweede Sprint nodig bleek.

5.2 Sprint 2

Tijdens de tweede Sprint hebben we ons voornamelijk de bij E.Novation gebruikte technologie eigen gemaakt. We hebben in deze Sprint ook een deel van het introductieprogramma voor nieuwe medewerkers van E.Novation gevolgd. We hebben daarbij iets mogen zien van de andere afdelingen: sales, de helpdesk en operations, de systeembeheerafdeling. Door deze introducties kregen we niet alleen een beter beeld van het bedrijf, maar ook van het EMS, door de verschillende facetten van het product die aan bod komen bij de verschillende afdelingen.

We hebben de ontwikkelomgeving ingericht, met alles wat daarbij komt kijken: IDE-instellingen, Maven, subversion, lokale virtual machines als development omgeving, en ons eigen Hudson project. Ook zijn we erin geslaagd de ems-statistics-collector en de ems-monitor-daemon te draaien en te packagen, zonder dat de rest van de EMS nodig was.

We zijn relatief lang bezig geweest met het inrichten van alle systemen, onder meer omdat we met bepaalde technologiën nog geen ervaring hadden. Voorbeelden hiervan zijn Maven, RPM scripts en E.Novation-specifieke tools.

5.3 Sprint 3

De statistics collector en monitor daemon-component zijn tijdens deze sprint samengevoegd in de E.Novation Monitor Daemon, enomon. Het heeft drie dagen gekost om alle configuratie-files hiervoor goed te zetten, samen te voegen en te testen.

Tijdens deze Sprint is ook het verzoek om een queue-check voor het product LSP Connect binnengekomen. Hierbij is gevraagd om ook de leeftijd van het bericht in een queue weer te geven, iets wat niet in de ems-statistics-collector zit. Tijdens deze Sprint is hiervoor de AgeCollector geschreven. Door op dit verzoek in te gaan, zijn andere stories, zoals het naast elkaar kunnen uitvoeren van enomon en de statistics collector, doorgeschoven naar de vierde Sprint.

De JMX- en leeftijdsdata wordt sinds deze Sprint als ‘passive check’ naar Nagios opgestuurd. Hiervoor is de subscribe-kant van de ems-statistics-collector volledig herschreven naar het systeem zoals beschreven in 4.1 en 4.2.

In de demo van deze Sprint hebben we onder meer de check op de leeftijd van het bericht in een queue in Nagios laten zien.

Deze Sprint is soepel verlopen. Hoewel de vraag van LSP Connect ons minder tijd liet voor andere, in het licht van onze oorspronkelijke opdracht, belangrijkere stories, was het een goede opsteker om het bedrijf te kunnen voorzien van iets dat vrij direct in gebruik zou worden genomen.

5.4 Sprint 4

Vanuit E.Novation komt aan het begin van deze Sprint de wens om de checks eenvoudiger configureerbaar te maken dan 20 regels XML en een configuratiebestand voor elke nieuwe check-instantie. Dit is het hoofddoel voor deze Sprint geworden. Daarnaast zijn we met wat kleinere zaken bezig geweest: het naast elkaar kunnen uitvoeren van enomon en de ems-statistics-collector.

We zijn dus vooral druk bezig geweest met het programmeren van het makkelijker configureren van de checks. Doordat sommige variabelen daarmee pas runtime beschikbaar zijn, moeten veel afhankelijkheden in eigen code ingesteld worden, in plaats van met behulp van Spring.

Bij het afronden van deze Sprint wordt alle configuratiecode nagelopen op ontwerp- en programmeerfouten. Hoewel er voortdurend codecontrole plaatsvind, was er hier specifiek tijd nodig vanwege de hoeveelheid geproduceerde code.

Het doel van deze Sprint blijkt niet gehaald te worden. De configuratiestory is niet af, en daarmee is de hoeveelheid afgeronde stories te klein om te presenteren. De story bleek veel meer werk dan we geschat hadden.

De configuratiestory is wel nodig, en dus doorgeschoven naar de volgende sprint, samen met de opdracht van LSP Connect. Tijdens deze Sprint is bleek uit de logfiles dat er door OpenMQ fouten werden gegeven bij het doen van meerdere aanvragen tegelijkertijd. Dit bleek een bekend probleem, en er wordt besloten hier in de volgende Sprint aan te gaan werken.

5.5 Sprint 5

Voor deze Sprint was het doel om naast de module voor LSP Connect ook een module voor Zorgmail op te leveren, en om de updates naar RRDTool werkend te krijgen. Eveneens was het doel om de functionaliteit van ems-monitor-daemon, het op aanvraag van Nagios kunnen checken, weer werkend te krijgen.

Bij het proefdraaien van de applicatie ontdekten we een performance probleem, waarvan we dezelfde dag concludeerden dat het waarschijnlijk bij JSendNSCA lag. Door het schrijven van de bulkverzending van Statussen hebben we dit opgelost.

Er is een workaround voor het concurrency probleem in OpenMQ geprogrammeerd in de vorm van het sequentieel uitvoeren van alle checks. Zie ook paragraaf 4.2.

In de loop van de Sprint blijkt er te weinig tijd te zijn om ook de volledige functionaliteit van de ems-monitor-daemon in enomon te herstellen. Het gaat hierbij om het kunnen aanroepen van checks in de monitor daemon vanuit Nagios. Het terugzetten hiervan is niet triviaal, doordat alle voor de ems-

monitor-daemon geconfigureerde checks geport en vooral getest moeten worden. Het tijdsgebrek ontstond door de ontdekte performance en OpenMQ problemen.

Tijdens de demo van deze Sprint hebben we aan E.Novation laten zien wat we in tien weken bij hen bereikt hebben: de nieuwe E.Novation Monitor Daemon, die ze voor Zorgmail en LSP Connect kunnen gaan gebruiken.

6 Evaluatie en aanbevelingen

Een van de belangrijkste aspecten aan onze stagedoelen was het opleveren van een systeem, dat geschikt was om in productie te worden genomen. Dit had een grote impact op onze productiviteit in termen van onze oorspronkelijke projectdoelen. Onze “definition of done” was veel stringenter dan bij een project waarbij bijvoorbeeld een proof-of-concept wordt ontwikkeld. Het was onacceptabel om een niet-werkend product op te leveren, zowel in termen van functionaliteit als in termen van fouten. Bovendien moesten tussenproducten en eindproduct kant-en-klaar uitrolbaar zijn in de productieomgeving. Aan de andere kant hebben we, door het Scrum raamwerk te gebruiken, wel een ontwikkelmethode gevolgd die daar goed geschikt voor is.

6.1 Beperking van de omvang van de opdracht

We hebben – in overleg met onze opdrachtgever – een duidelijke keuze gemaakt voor kwaliteit over kwantiteit. We hebben de scope van het project aan het begin van de vierde sprint beperkt tot wat we uiteindelijk opgeleverd hebben: de oorspronkelijke doelen waren erg ambitieus gezien de verschillende randvoorwaarden waarbinnen we moesten opereren.

6.1.1 Technische problemen

Tijdens ons project zijn we tegen twee belangrijke technisch georiënteerde problemen gelopen. De eerste is de complexiteit van de ontwikkelomgeving. De tweede gaat meer om de technische context van de opdracht c.q. het op te leveren product.

6.1.1.1 Ontwikkelomgeving

Hoewel we beiden enige ervaring hadden met het werken met een geïntegreerde ontwikkelomgeving, is het toch wel wat anders als die vervolgens een onderdeel wordt van een compleet traject van ontwikkeling, inclusief uitrol naar een (ontwikkel)server. De software artefacten die opgeleverd moesten worden, waren daarom technisch complexer dan waar we ervaring mee hadden. Het leren werken met deze technische omgeving zorgde voor aanzienlijke vertraging in de tweede en derde sprint.

6.1.1.2 Bestaand systeem en het EMS

Een tweede aspect dat het ontwikkelwerk complex maakte was het feit dat we ons nieuwe product moesten integreren in een bestaande omgeving waar we geen ervaring mee hadden. Dit geldt voor de verschillende gebruikte technologieën (bijvoorbeeld Nagios en RRDtool) maar ook simpelweg voor de netwerkomgeving van E.Novation.

6.1.1.3 Suggesties

In het begin was voor ons de grootste drempel voor effectief werken het gebrek aan kennis ten aanzien van de verschillende gereedschappen en de technische omgeving waarin we terecht kwamen. Om toekomstige stageprojecten soepeler te laten verlopen en effectiever te laten zijn, hebben we voor E.Novation de volgende suggesties (in aflopende volgorde van complexiteit):

1. Maak een handleiding voor nieuwe ontwikkelaars, met daarin gedocumenteerd alle relevante kennis die een nieuwe ontwikkelaar nodig heeft. Onder andere de hieronder volgende suggesties kunnen daarin worden opgenomen.

Dit is een zeer grote investering, en waarschijnlijk is dit te veel gevraagd. Zo lang de doorstroom van ontwikkelaars niet zo groot is, is dit niet heel erg nodig. Bovendien veranderen waarschijnlijk een hoop zaken vaak, waardoor het ook nog eens veel werk is om het document te onderhouden.

2. Maak een gestructureerd draaiboek voor mondelinge kennisoverdracht van ervaren naar nieuwe ontwikkelaars. Door dit te structureren is het verhaal duidelijk, kan de nieuwe ontwikkelaar e.e.a. nog eens teruglezen, en wordt niets vergeten.
   Dit idee is wel interessant, omdat hier toch al over nagedacht moet worden: de kennisoverdracht moet überhaupt plaatsvinden. Bovendien hoeft dit geen directe grote investering te zijn: het kan gaandeweg worden geïmplementeerd en geperfectioneerd.
3. Schrijf een complete walkthrough voor het toegang verkrijgen tot alle onderdelen van het ontwikkeltraject: van het uitchecken van de source code tot de installatie van RPMs op de verschillende ontwikkelservers.
   Dit is waarschijnlijk niet veel werk, maar wel heel erg nuttig. Zo krijgen nieuwe ontwikkelaars/stagairs direct een duidelijk beeld van wat er allemaal komt kijken bij het opleveren van artefacts: omdat dit onderdeel is van de “definition of done”, is het belangrijk dat iedere ontwikkelaar hier zo snel mogelijk bekend mee is.
4. Zet op de interne wiki een overzicht van alle (voor het project) relevante services: Waar vind je SVN repositories? Op welke server draaien welke applicaties? Enzovoort.
   Dit laatste punt is erg praktisch, maar minder belangrijk, omdat dit door collega’s te vragen ook wel duidelijk wordt. Bovendien kan dit overzicht wellicht juist door nieuwe ontwikkelaars aangelegd worden, terwijl ze dit uitzoeken.

Wij raden E.Novation aan om in ieder geval serieus naar punt 2 en 3 te kijken. Een aardig deel van de benodigde informatie is al op de interne wiki aanwezig, maar hier moet ook nog flink op worden uitgebreid. Het belangrijkste is een centraal punt van waar al deze informatie in hapklare brokken te vinden is.

6.1.2 Beperkte teamgrootte: twee personen

Een andere belangrijke factor die ons beperkt heeft in de hoeveelheid werk die we hebben uitgevoerd, is dat we slechts met twee personen de stageopdracht hebben uitgevoerd. Het Scrum proces is het effectiefst als er met teams van rond de zeven personen wordt gewerkt. Binnen het Bachelor Project Technische Informatica van de TU Delft wordt met twee tot vier personen gewerkt. Het is aan te raden bij volgende projecten met zeker vier personen een team te vormen. Naast het feit dat een groter team überhaupt meer werk kan verrichten, kan er ook binnen het team meer gespecialiseerd worden (zie ook paragraaf 3.1.1 over het Scrum ontwikkelteam). Tot slot is het voor de stagairs interessanter om in een groter team te werken vanwege de uitdagingen op management gebied – maar het Scrum model is hiervoor uitstekend ingericht, dus dat moet lukken.

Hier is overigens ook voor de TU een interessant probleem: het Bachelor Project wordt momenteel met maximaal vier personen uitgevoerd. Maar software bouwen wordt vaak ook in grotere teams gedaan. Het is wellicht een idee om, binnen op Scrum gebaseerde projecten, met grotere teams te werken, afhankelijk van de wensen van het stagebedrijf.

6.1.3 Te veel rollen voor onze stagebegeleider

Tijdens onze stage hadden we één directe begeleider, Bastiaan Bakker. Hij vervulde een aantal rollen, die niet altijd even goed verenigbaar zijn. Bovendien kostte dit hem veel tijd. Ten eerste was hij Scrum Master en Product Owner tegelijkertijd. Verder heeft hij kwaliteitscontrole uitgevoerd op functioneel niveau. Tot slot heeft hij met enige regelmaat meegedacht over de technische implementatie van het product.

We hadden af en toe een remming op onze voortgang doordat onze stagebegeleider geen tijd had om een van zijn rollen te vervullen (zie ook paragraaf 3.3, evaluatie toepassing Scrum). Dit hebben we – samen met onze begeleider – pas laat in de stage onderkend, op een punt dat daar eigenlijk niet veel meer aan gedaan kon worden. Het is belangrijk om hier bij nieuwe projecten goed op te letten.

Een oplossing kan zijn dat de QA rol en de technische ondersteuningsrol expliciet ingevuld worden door andere medewerkers van E.Novation. Door niet slechts van een persoon afhankelijk te zijn, is het makkelijker om tempo te houden.

6.2 Evaluatie projectdoelen

De doelen die aan het begin van de stage zijn gesteld, zijn gedeeltelijk bereikt. We gaan per doel in op de mate waarin eraan voldaan is.

6.2.1 Samenvoeging en modularisatie

Het doel was om de monitor daemon en statistics collector samen te voegen in één applicatie, die onafhankelijk zou zijn van de rest van de EMS. De opgeleverde Enomon voldoet hieraan. Deze biedt één punt om controles op diensten en het verzamelen van statistiek over diensten uit te voeren.

De Enomon is modulair opgezet, met een basis en systeemspecifieke modules voor Zorgmail en LSP Connect. Hierdoor is niet alleen de code onafhankelijk, maar is ook de hele applicatie breder inzetbaar.

6.2.2 Functionaliteit statistics collector

De statistics collector kon statistiek verzamelen van algemene JMX-diensten, van JVM geheugen en van Oracle Connection caches. Bij het ontwikkelen van de Enomon is de aandacht voornamelijk uitgegaan naar de JMX-diensten. De andere statistiekbronnen zijn hierdoor waarschijnlijk onbruikbaar geworden: hoewel de code wel gecompileerd kan worden, is deze verder niet getest. De nieuwe configuratie is beschikbaar voor statistiek op JMX-diensten en de leeftijd van een bericht in een wachtrij in de EMS.

6.2.3 Functionaliteit monitor daemon

Het doel was om het systeem van polling door Nagios te converteren naar een event-driven model. De checks in Enomon sturen event-driven resultaten voor JMX-diensten en berichtleeftijden naar de Nagios server. Er zijn daarmee duidelijk minder soorten checks op EMS-diensten mogelijk dan in de oude monitor daemon. De polling mogelijkheid zoals die bestond is niet in de Enomon geïntegreerd.

Één van de checks die nog niet geïmplementeerd zijn is de mailloop, waarbij een bericht door de EMS gestuurd wordt om de integrale werking van het systeem te testen. Hier zijn we niet aan

toegekomen, doordat de prioriteit uiteindelijk bij de LSP Connect-module en eenvoudigere configuratie lag. De implementatie hiervan is dermate omvangrijk dat hier een volgende stage aan besteed zou kunnen worden.

6.3 Kennismaking met Scrum

Tijdens onze stage hebben we voor het eerst kennisgemaakt met Scrum, in de theorie, maar vooral ook in de praktijk. Het bleek een bijzonder effectieve manier van software ontwikkeling te zijn door de focus op toegevoegde waarde versus 'afval'. Dit was in eerste instantie wel erg wennen. In de Bachelor opleiding Technische Informatica ligt de focus tijdens projecten veel meer op technologie, ontwerp, documentatie en proces. De stage heeft ons dan ook een belangrijke nieuwe vaardigheid bijgebracht: Het kunnen richten op toegevoegde waarde voor de klant, en het inspelen op veranderende wensen.

Voorafgaand aan onze stage hadden we dus nog helemaal geen ervaring met Scrum. Dat zorgde ervoor dat we pas gaandeweg een aantal aspecten begonnen te begrijpen. En sommige dingen zijn we pas echt goed gaan begrijpen tijdens het schrijven van dit projectverslag: door te reflecteren is ons begrip van het Scrum raamwerk groter geworden. In hoofdstuk 3 evalueren we de door ons toegepaste implementatie van het Scrum proces uitgebreid.

7 Bibliografie

Beck, K., Beedle, M., Bennekum, v. A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). *Manifesto for Agile Software Development*. Opgeroepen op July 13, 2010, van http://www.agilemanifesto.org/

Eugster, P. T. (2003, June). The many faces of publish/subscribe. *ACM Comput. Surv., 35*(2), 114-131.

Grenning, J. (sd). *Planning Poker.* Opgeroepen op Juli 19, 2010, van Rennaissance Software: http://renaissancesoftware.net/papers/14-papers/44-planing-poker.html

Leffingwell, D. (2007). Twelve key principles of XP. In D. Leffingwell, *Scaling Software Agility - Best Practices for Large Enterprises* (pp. 35-38). Crawfordsville, California: Pearson Education, Inc.

Levison, M. (2008, December 11). *InfoQ: Can Product Owner and Scrum Master be combined?* Opgeroepen op 07 15, 2010, van InfoQ: Tracking change and innovation in the software development community: http://www.infoq.com/news/2008/12/scrum-master-product-owner

Patton, J. (2010, Oktober 8). *The new user story backlog is a map*. Opgeroepen op Juli 19, 2010, van Jeff Patton's Holistic Product Design & Development Blog: http://www.agileproductdesign.com/blog/the_new_backlog.html

Schwaber, K., & Sutherland, J. (2009). *Scrum Guide.* Retrieved June 6, 2010, from Scrum.org - The Home of scrum: http://www.scrum.org/scrumguides/

Wells, D. (sd). Opgeroepen op Juli 21, 2010, van Extreme Programming: A gentle introduction: http://www.extremeprogramming.org/

Appendix A Agile software ontwikkeling

De afdeling Application Development bij E.Novation werkt met het Scrum raamwerk, een agile ontwikkelmodel, als basis voor het ontwikkelproces. Tijdens onze stage zullen wij een implementatie van Scrum gebruiken. Daarom hebben we onderzocht hoe softwareontwikkeling met behulp van het Scrum raamwerk er uit ziet. In paragraaf A.1 beschrijven we wat 'agile' softwareontwikkeling inhoud. We hebben onze bevindingen over Scrum beschreven in paragraaf A.2. Scrum is een raamwerk voor het beheersen van een proces. Het geeft echter geen invulling aan een aantal concrete zaken op ontwikkelgebied. Dit gat wordt heel vaak, en ook bij E.Novation, gevuld met best practices uit de Extreme Programming methodiek. Daarom hebben we hier ook een kleine studie naar gemaakt. Een aantal onderdelen van XP hebben we beschreven in paragraaf A.3.

A.1 Wat is agile?

Voordat we het Scrum model en de Extreme Programming aspecten bespreken, geven we eerst een korte introductie tot de concepten van agile software ontwikkeling. Dit doen we aan de hand van het Agile Manifesto (Beck, et al., 2001). Dit is een samenvatting van de gedachten die ten grondslag liggen aan de verschillende agile ontwikkelmethoden die in de loop der jaren gepubliceerd zijn:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions*** *over processes and tools*
- ***Working software*** *over comprehensive documentation*
- ***Customer collaboration*** *over contract negotiation*
- ***Responding to change*** *over following a plan*

That is, while there is value in the items on the right, we value the items on the left more."

Dit alles is voortgekomen uit de realisatie dat het traditionele watervalmodel van softwareontwikkeling niet effectief en efficiënt was. Veel projecten gingen over budget in termen van tijd en geld. Bovendien leverden ze vaak ook niet een eindproduct op waar de klant tevreden mee was. Meestal had dit te maken met veranderende wensen van de klant. Tijdens de Requirements Analysis fase van het watervalmodel werd tot in het kleinste detail vastgelegd wat gebouwd moest gaan worden. Vervolgens werd dan het product gebouwd. Maar enerzijds wist de klant eigenlijk nog niet precies wat hij daadwerkelijk wilde – pas zodra er een werkend systeem was kon hij aangeven wat daar mis mee was. Anderzijds veranderden de requirements simpelweg vanuit de bedrijvigheid: de klant had in de tijd tussen het vastleggen van requirements en het opleveren van het eindproduct zijn bedrijfsproces aangepast.

Om deze problemen op te lossen, zijn in de loop der jaren een aantal verschillende nieuwe methoden van ontwikkelen bedacht. De essentie van agile ontwikkelen ligt dus in de focus op het opleveren van werkende software, die voldoet aan de wensen van de klant. Dat betekent dat flexibel moet worden omgegaan met veranderende wensen van de klant; het ontwikkelproces moet verandering ondersteunen en omarmen.

A.2 Het Scrum raamwerk

Scrum is een raamwerk dat voor het eerst beschreven is door Jeff Sutherland en Ken Schwaber, en een uitgebreide introductie is te vinden in (Schwaber & Sutherland, 2009).

Het Scrum raamwerk is volgens Schwaber en Sutherland een empirische proces controle methode, gebaseerd op drie pilaren: transparantie, inspectie en aanpassing. Transparantie betekent dat alle aspecten die op het eindresultaat van het (Scrum) proces van toepassing zijn, zichtbaar moeten worden gemaakt. Door die aspecten te inspecteren – wordt het optimale resultaat opgeleverd? – kan worden bepaald of bepaalde procesonderdelen moeten worden aangepast. Door continue aanpassing van het proces wordt het eindresultaat geoptimaliseerd.

Er zijn in het Scrum raamwerk drie momenten waarop inspectie en aanpassing plaats kan vinden. Tijdens de Daily Scrum Meeting wordt de voortgang binnen de lopende Sprint gecontroleerd. Hier worden aanpassingen gemaakt aan het proces ten behoeve van de waarde van de eerstvolgende werkdag (tot de volgende DSM). Tijdens de Sprint Review wordt het product release doel geëvalueerd en worden aanpassingen gemaakt ten behoeve van waarde optimalisatie van de volgende Sprint. Ten slotte wordt tijdens de Sprint Retrospective de afgelopen Sprint besproken. De Sprint Retrospective wordt gebruikt om de volgende Sprint productiever te maken.

Binnen het Scrum raamwerk wordt in Scrum Teams gewerkt aan de realisatie van de ontwikkeling van producten. Dit kunnen nieuwe producten zijn, maar ook uitbreidingen, vernieuwingen, of andersoortige aanpassingen aan bestaande producten. Bovendien hoeft het niet per se om softwareontwikkeling te gaan: Scrum kan ook in andere vakgebieden toegepast worden. In alle gevallen wordt door middel van incrementele ontwikkeling een eindproduct opgeleverd.

A.2.1 Het Scrum Team

Binnen het Scrum raamwerk wordt gewerkt in Scrum Teams. Deze teams zijn cross-functional en self-organizing. Binnen deze teams zijn drie rollen gedefinieerd. Allereerst is er de Scrum Master, die verantwoordelijk is voor het optimaliseren van het Scrum proces. Ten tweede is er de Product Owner, die ervoor moet zorgen dat de waarde van het werk dat het Scrum Team doet, gemaximaliseerd wordt. Ten derde is er het Team, een groep van ontwikkelaars die het daadwerkelijke implementatiewerk doen.

A.2.1.1 Scrum Master

De Scrum Master faciliteert het gehele Scrumproces. Dit doet hij door alle obstakels die het team in de weg liggen om de doelen van een sprint te halen, weg te nemen. Maar het is ook zijn taak om te letten op afgesproken regels, zoals: schrijven Teamleden gemaakte uren, zodat het Scrum Team de voortgang tijdens de sprint goed in de gaten kan houden? De Scrum Master is dus degene die zorgt dat het werktempo van het team optimaal blijft.

A.2.1.2 Product Owner

De Product Owner is gedurende het gehele traject van productontwikkeling de stem van de klant. Hij is degene die bepaalt wat en wanneer uiteindelijk als (eind)product wordt opgeleverd. Hij ontwikkelt een visie op het project of product, definieert de doelen op de langere termijn (over meerdere Sprints heen), en vult de Product Backlog met wensen van de klant.

Vervolgens zal de Product Owner prioriteiten toekennen aan de verschillende items in de backlog. Dit doet hij op basis van de waarde die iedere wens toevoegt aan het product voor de klant. Die waarde kan hij bepalen door te kijken naar de waarde in financiële termen, ten opzichte van de tijd die het zal kosten om die functionaliteit te implementeren. Maar ook andere vormen van waardebepaling en prioritering zijn goed denkbaar: sommige items zullen kritisch zijn voor het bedrijfsproces, terwijl andere dat minder kunnen zijn. Door een combinatie van deze methoden te gebruiken, komt de Product Owner tot een geprioriteerd backlog, welke het team zal gaan implementeren.

Een van de belangrijkste aspecten aan de rol van Product Owner is dus dat hij mandaat moet hebben binnen de organisatie om beslissingen te nemen: over welke items uit de Product Backlog welke prioriteit krijgen, over welke kwesties (nieuwe features of bugs) in de Product Backlog worden opgenomen, enz.

A.2.1.3 (Ontwikkel)team

Het omzetten van de product backlog in incrementeel verbeterde productiteraties is het hoofddoel van het ontwikkelteam. Dit team is normaliter 'cross-functional': teamleden met verschillende vaardigheden worden bij elkaar gezocht om een product of project te ontwikkelen. Voor software engineering betekent dit, dat vaardigheden op alle gebieden binnen die discipline samenkomen in één team. Denk hierbij onder meer aan requirements analyse, architectuur ontwerp, technisch ontwerp, implementatie & unit testing, documentatie (zowel technisch als voor de eindgebruiker), integration testing, deployment op testomgevingen, acceptance testing en deployment naar productieomgevingen. Zonder al deze vaardigheden is het (vaak) niet mogelijk om een compleet werkend (eind)product op te leveren, en zal dus een release nooit lukken.

Teams bestaan gewoonlijk uit 5 tot 9 leden. Om efficiënte communicatie mogelijk te houden is een beperking aan de omvang wenselijk. Bij weinig teamleden zal de toegevoegde waarde van extra communicatie over en weer tussen teamleden minimaal zijn. Bovendien zal het beschikbare aantal vaardigheden beperkt zijn. Bij een beperking in vaardigheden kan het voorkomen dat het Team niet in staat is om een release daadwerkelijk te doen.

Het ontwikkelteam werkt op basis van de Product Backlog, zoals die is opgesteld door de Product Owner. Aan het begin van een sprint wordt bepaald wat in de Backlog voor die specifieke Sprint wordt opgenomen, en vervolgens gaat het team de betreffende Backlog items implementeren. Het team organiseert hierbij zichzelf: de leden bepalen zelf aan welke items ze werken, op basis van de prioriteiten in de backlog.

De rollen van Scrum Master en Product Owner kunnen eventueel door leden van het Team ingevuld worden, maar kunnen beter niet beide door dezelfde persoon worden vervuld: ze hebben tegenstrijdige belangen. De Product Owner richt zich op (nieuwe) ideeën voor functionaliteit – de Product Backlog – en wil dit binnen zo kort mogelijke tijd uitgevoerd zien. De Scrum Master richt zich op het implementeren van die functionaliteit in een tempo dat optimaal is – dus niet maximaal. Vanwege die tegenstrijdigheid is het goed om die rollen gescheiden te houden. Bovendien brengen beide rollen taken met zich mee, die veel tijd kosten en daardoor ook niet goed te combineren zijn (Levison, 2008).

A.2.2 Inhoudelijke richting: de Product Backlog

Om richting te geven aan het ontwikkelproces wordt bij Scrum gebruik gemaakt van een Product Backlog, waarin alle nog te ontwikkelen functionaliteit wordt opgenomen. De Product Backlog wordt geprioriteerd door de Product Owner en vervolgens wordt dan in de geprioriteerde volgorde ontwikkeld door het Team.

In de Product Backlog kunnen in principe allerlei vormen van functionele requirements worden opgenomen: soms is het praktisch om een Use Case op te nemen, maar meestal wordt met User Stories gewerkt (zie paragraaf A.3.1).

A.2.2.1 Prioritering van de Backlog met focus op toegevoegde waarde

Ieder item in de Product Backlog creëert toegevoegde waarde. De items met de meeste waarde krijgen de hoogste prioriteit. Hierdoor wordt gegarandeerd dat de gewenste features of aanpassingen geïmplementeerd worden in een volgorde waarbij zo snel mogelijk zo veel mogelijk waarde wordt gecreëerd.

A.2.3 Management van het ontwikkelproces

Het ontwikkelproces wordt binnen Scrum beheerst door middel van verschillende time-boxes. Om te beginnen is er de Release Planning Meeting. Vervolgens is er de Sprint, een vaste periode van ontwikkeling, waarin een uitbreiding of aanpassing van een product wordt gerealiseerd. Tijdens de Sprint zijn er de Sprint Planning Meeting, de Sprint Review, de Sprint Retrospective, en dagelijks de Daily Scrum Meeting.

A.2.3.1 Release Planning Meeting

Tijdens de Release Planning Meeting worden de doelen voor het project of product vastgelegd in termen die voor alle betrokkenen duidelijk zijn: het Scrum Team, maar ook de rest van de organisatie. Het Release Plan wordt gemaakt, waarin doel van de release, hoogste prioriteit Product Backlog, grootste project risico's en een high-level overzicht van de gewenste features en functionaliteit van het product worden vastgelegd. Als dit vervolgens gedaan is, kan met de eerste Sprint worden begonnen.

A.2.3.2 Sprints

Het werk aan de software wordt opgedeeld in Sprints: iteraties van twee tot vier weken, waarna een release wordt opgeleverd. Tijdens een sprint wordt gewerkt aan een deel van de Product Backlog, dat vooraf wordt vastgesteld. Sprints hebben een vaste lengte. Ze worden dus niet verlengd als werk dreigt uit te lopen.

Een sprint wordt begonnen met een Sprint Planning, waarbij het plan voor die Sprint wordt vastgelegd. Aan het eind van de sprint worden een Sprint Review en een Sprint Retrospective gehouden.

Tijdens de Sprint wordt het daadwerkelijke implementatiewerk uitgevoerd. Daarbij wordt de "definition of done" altijd gebruikt als maatstaf om te bepalen of de implementatie van een item uit de Backlog compleet geïmplementeerd is. De definition of done bevat alle eisen op dat gebied: bijvoorbeeld dat er altijd unit tests voor geschreven code moeten zijn, dat er bepaalde documentatie

stukken moeten zijn geschreven, enzovoort. Door ieder team wordt de definition of done anders ingevuld.

A.2.3.3 Sprint Planning Meeting

De Sprint begint met het maken van een Sprint planning: de Sprint Backlog, met de items uit de Product Backlog die in die Sprint gaan worden geïmplementeerd. Hiervoor moet een aantal zaken worden vastgelegd: de wensen van de klant op de korte termijn, de kosten van de implementatie van die wensen (uitgedrukt in tijd), en de prioriteit die de betreffende wensen hebben.

Het ontwikkelteam maakt een inschatting van die kosten. Dit kan bijvoorbeeld gedaan worden door middel van Planning Poker (zie paragraaf A.3.1.1).

Op basis van de inschattingen die het ontwikkelteam heeft gedaan, kan de Product Owner vervolgens bepalen wat de uiteindelijke prioriteiten voor de Sprint zullen worden. Samen met het ontwikkelteam bepaalt hij vervolgens welke User Stories daadwerkelijk in de Sprint Backlog terecht zullen komen: De Sprint zal een vaste tijd duren, en het ontwikkelteam legt zich dus vast op het implementeren van de Sprint Backlog binnen de Sprint.

A.2.3.4 Sprint Review

Aan het eind van de Sprint wordt een Sprint Review (of Sprint Demo) gehouden. Hierbij zijn alle stakeholders aanwezig. Er wordt besproken wat in de afgelopen Sprint ontwikkeld is, en wat niet. Ook wordt een een demonstratie gegeven van de geïmplementeerde functionaliteit. Vervolgens wordt de Product Backlog besproken. Op basis daarvan wordt bepaald waar de prioriteiten voor de volgende Sprint liggen.

A.2.3.5 Sprint Retrospective

De Sprint Retrospective is hét moment binnen Scrum om te evalueren en verbeterpunten te vinden. Daarmee is de Sprint Retrospective ook een heel belangrijk moment in het Scrum proces. Tijdens de rest van de Sprint ligt de focus op het ontwikkelen van het product, en is er in principe geen ruimte om te terug te kijken op het verloop van het proces. Dat gebeurt tijdens de Sprint Retrospective juist wel.

A.2.3.6 Daily Scrum Meeting

Iedere dag wordt, meestal aan het begin van de dag, een kort overleg gehouden waarin besproken wordt, wat er sinds de vorige DSM door de teamleden gedaan is en wat er tot de volgende DSM gedaan gaat worden. Bovendien, en vooral, wordt besproken of er problemen verwacht worden bij de implementatie van de User Stories die voor de direct komende tijd op het programma staan. Als dit zo is, dan is het dus aan de Scrum Master om die ‘impediments’ weg te nemen.

Bij dit overleg zijn alle teamleden, inclusief Scrum Master en Product Owner, aanwezig. Eventueel kunnen ook andere stakeholders hierbij aanwezig zijn, maar die hebben in principe geen spreekrecht. De DSM is dus puur bedoeld om binnen het team te bespreken hoe de voortgang is. Het is expliciet niet de bedoeling dat buitenstaanders zich gaan bemoeien met de interne organisatie van het team!

A.3 Extreme Programming best practices

Extreme Programming is op zichzelf een model voor softwareontwikkeling. Maar we zijn vooral geïnteresseerd in een aantal best practices die vaak binnen het Scrum raamwerk (en andere ontwikkelmethoden) worden toegepast. De volgende concepten zullen we behandelen: User Stories, Story of Task Boards, Test-Driven Development, Continuous Integration, Incremental Design en Research Spikes. Deze worden grotendeels beschreven in (Leffingwell, 2007). Ook is (Wells) een goede bron van informatie over Extreme Programming.

A.3.1 User Stories

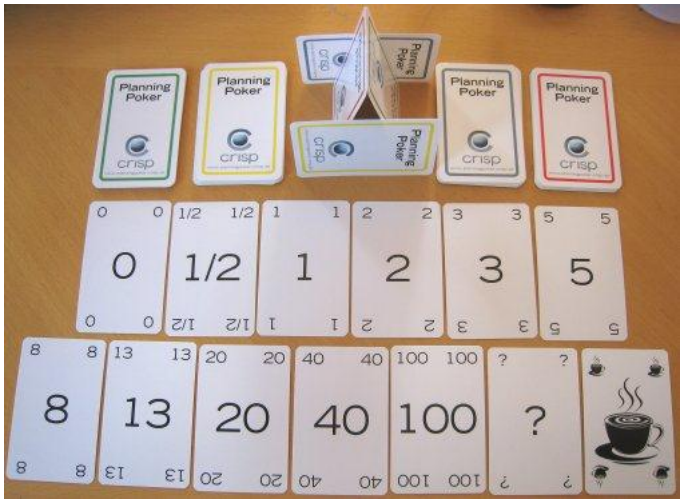
Om de te ontwikkelen functionaliteit van het systeem in behapbare stukken te krijgen, wordt dit zo ver mogelijk uitgesplitst. Ieder stukje functionaliteit, dat niet meer verder opgesplitst kan worden, vormt dan een User Story. Iedere User Story beschrijft globaal een stukje van het te ontwikkelen systeem in termen van de actie die een gebruiker ('User') wil doen, het doel wat hij wil bereiken, en de reden voor dat doel. Essentieel is hierbij in te zien dat het erom gaat de kleinst mogelijke User Stories te vinden die nog steeds toegevoegde waarde creëren voor de klant.

De acceptatiecriteria van de Story worden vervolgens met behulp van de klant of Product Owner opgesteld, zodat duidelijk gedefinieerd wordt, waaraan de ontwikkelde software ten aanzien van de nieuwe functionaliteit moet voldoen.

Voor de implementatie van iedere User Story moeten werk gedaan worden. Dit werk wordt opgedeeld in Tasks, die gekoppeld blijven aan de specifieke Story. Tasks worden opgesteld door het ontwikkelteam, en worden ook vanuit het perspectief van de ontwikkelaars opgesteld. Het gaat bij Tasks dus niet om waarde voor de klant, maar om implementatie van een Story. De klant krijgt dus in principe ook niet te maken met Tasks (anders dan dat ze op het Story of Task Board kunnen voorkomen, zie paragraaf A.3.2).

A.3.1.1 Schatten van kosten User Stories

Bepaling van de waarde van een User Story kan pas als ook duidelijk is hoeveel het gaat kosten om die User Story te implementeren. Daarom wordt voor een nieuwe Sprint eerst van de relevante User Stories bepaald hoeveel tijd het gaat kosten om ze te implementeren. In eerste instantie worden de User Stories met Story Points geschat: hoe moeilijk is het en hoeveel tijd gaat het kosten om deze Story te implementeren? De Story Points geven dan aan hoe kostbaar de betreffende Story relatief is ten opzichte van andere Stories.



Figuur 13: Een planning poker kaartenset

Deze inschatting wordt gedaan met behulp van planning poker door het ontwikkelteam (Grenning). Hierbij wordt per teamlid een kaartenset gebruikt (zie Figuur 13). Iedere Story, die in de Product Backlog terecht is gekomen, en nog niet een inschatting heeft, wordt vervolgens beoordeeld. Daarbij kiest ieder teamlid, uit een gesloten hand, een kaart met een bepaalde score. Tegelijkertijd leggen dan alle teamleden hun gekozen kaart op tafel open, zodat de teamleden elkaar niet kunnen beïnvloeden bij hun keuze. Vervolgens wordt, als de inschattingen van elkaar verschillen, een korte discussie gevoerd over de betreffende Story: zijn de teamleden het eens over de scope en de inhoud van de Story? Op die manier moet het team dan gezamenlijk tot een goede inschatting van de Story Points komen. Overigens moeten ook Stories die meer dan 20 punten krijgen, nog eens goed bekeken worden: dit zijn meestal kandidaten om opgesplitst te worden.

Ten slotte wordt, op basis van ervaringen uit het verleden, bepaald hoeveel tijd de implementatie van de verschillende Stories gaat kosten. Met die informatie kan de Product Owner dan aan de slag bij het prioriteren van de Product Backlog.

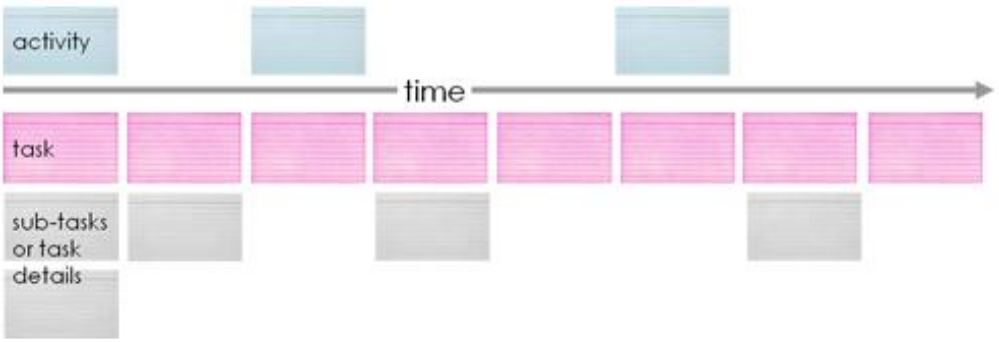
A.3.1.2 Hoe worden Stories ontdekt?

Er zijn verschillende manieren waarop een User Story tot stand kan komen. In eerste instantie zal de Product Owner bij het uiteenzetten van de doelen voor het product de wensen van de klant in kaart brengen, bijvoorbeeld door middel van Story Mapping . Een belangrijk deel van de Stories komt op deze manier tot stand. Maar ook en vooral tijdens de duur van een ontwikkeltraject zal de Product Owner tot nieuwe User Stories komen. Dit is precies waar agile ontwikkelmethoden hun kracht tonen: als de klant een nieuwe wens heeft, die heel belangrijk blijkt te zijn, dan kan die wens via een hoge prioriteit in de Product Backlog zijn weg vinden tot implementatie in de eerstvolgende Sprint.

Een andere manier waarop de Product Backlog uitgebreid kan worden, is dat tijdens een Sprint de Product Owner en/of het Team tot de conclusie komen dat er een essentieel stuk functionaliteit ontbreekt. Dit kan dan ook aan de Product Backlog worden toegevoegd, of zelfs aan de Sprint Backlog – maar dan zal een andere User Story uit de Sprint Backlog moeten worden verwijderd.

A.3.1.3 Story mapping (heading 4)

User Story Mapping is een gestructureerde techniek die helpt bij het in kaart brengen van de wensen van de klant ten behoeve van agile ontwikkelprocessen. Waar normaliter een eendimensionaal, plat Product Backlog wordt gebruikt (vaak in digitale vorm), wordt bij Story Mapping een groot stuk muur gebruikt om in twee dimensies een overzicht te geven van de gewenste functionaliteit van het eindproduct. Story Mapping wordt in deze vorm voor het eerst beschreven door Jeff Patton op zijn weblog (Patton, 2010).



Figuur 14: Abstracte Story Map

Figuur 14 is een voorbeeld van een abstracte Story Map. De gebruikte termen zijn geïntroduceerd door Patton. Een activity staat voor een overkoepelende User Story of Epic. De Epics staan geordend naar tijd: hoe lopen de verschillende gebruikers van de software door een systeem als ze verschillende, opeenvolgende acties wil ondernemen? Een voorbeeld hiervan is software voor een webshop. De verschillende activities kunnen dan zijn:

1. (Webshop) klant logt in voor een (virtueel) winkelmandje.
2. Klant maakt keuze en voegt een of meer artikelen toe aan winkelmandje.
3. Klant rekent artikelen af.
4. Webshopsysteem stuurt een bevestigingse-mail naar de klant.
5. Inventarissysteem maakt een verwerkingsorder aan voor magazijnmedewerkers.
6. Facturatiesysteem maakt een factuur aan voor de klant.
7. Enzovoort …

De activities worden vervolgens onderverdeeld in tasks: kleine User Stories die een specifieke actie van een gebruiker beschrijven. Daaronder hangen sub-tasks die helpen bij het uitvoeren van een task, bijvoorbeeld door verdere automatisering.

De (sub)tasks worden verticaal geordend op prioriteit. Op die manier kan een “walking skeleton” worden geïdentificeerd dat de eerste versie van het product kan gaan vormen: het meest minimale product dat releasebaar is. Vervolgens kunnen per iteratie de Stories gekozen worden om te implementeren, die de hoogste prioriteit hebben.

Om te controleren of het ontwikkelteam een “compleet” beeld van het systeem heeft, kan de Story Map gebruikt worden tijdens gesprekken met de klant. Door over de ‘tijdlijn’ met de klant de Map te

bespreken, kan hij goed aangeven wat belangrijk is, en wat minder belangrijk is, waar wensen ontbreken, en waar wensen overbodig zijn. Bovendien kan de Story Map makkelijk aangepast worden, zodra de klant aangeeft dat zijn wensen veranderen.

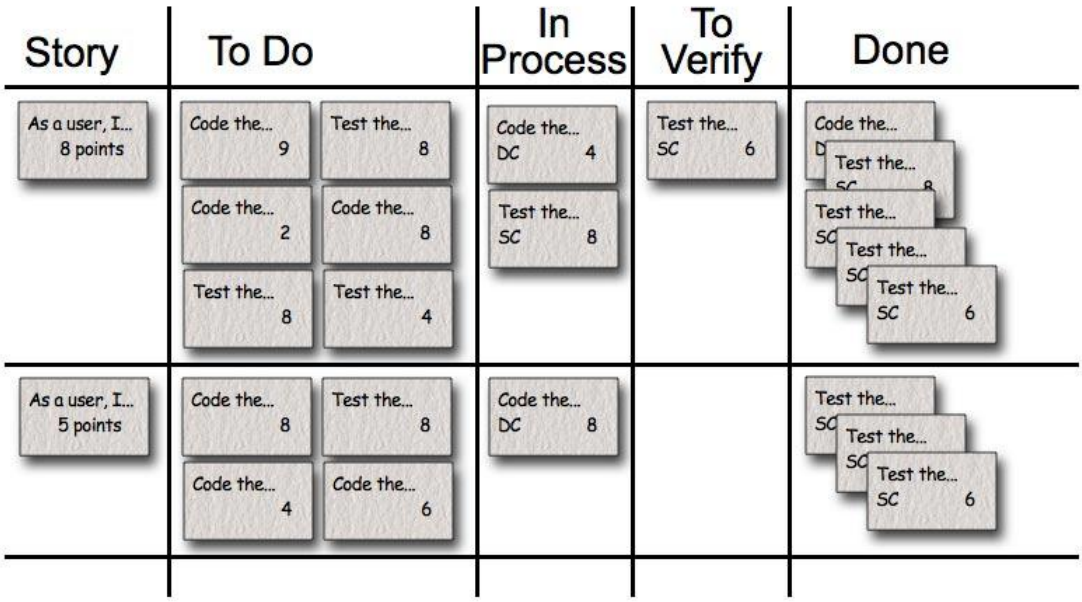
A.3.2 Story of Task Board

De voortgang binnen een Sprint wordt bijgehouden op een Story Board of Task Board. Dit is een stuk muur of een whiteboard, waarop gestructureerd bij wordt gehouden hoe de voortgang is van de implementatie van verschillende User Stories.

Figuur 15: Story Board tijdens stage (Oriëntatie fase)

In Figuur 15 is een voorbeeld van een Story Board te zien, zoals wij dat tijdens onze stage hebben gebruikt. Rechts hangt in de kolom TODO de Sprint Backlog. Vervolgens hangen in de kolom DOING de Stories waar op dat moment aan gewerkt wordt. In de kolom VERIFY hangen Stories die getest (alleen acceptance testing, alle andere vormen van testing worden eerder gedaan) of geverifieerd moeten worden. In de kolom DONE hangen de Sprint Backlog items die compleet afgerond zijn.

Sommige ontwikkelteams kiezen ervoor om niet alleen de Stories, maar ook de tasks op hun Task Board te tonen. Meestal zijn de Stories dan statische elementen op het Task Board, terwijl de Tasks verplaatst worden over het board naar mate in een volgende fase van ontwikkeling terecht komen (DOING, VERIFY, etc.). In Figuur 16 is zo'n Task Board te zien.



Figuur 16: Een mockup van een Task Board uit het Eclipse Process Framework

A.3.3 Incremental Design

Naar mate het product verder ontwikkeld wordt, wordt duidelijk hoe het ontwerp en de architectuur van het product eruit gaan zien. In feite wordt er voor iedere User Story, die geïmplementeerd moet worden, een stukje softwareontwerp uitgevoerd: wat moet er veranderd worden en hoe gaat dat er dan uit zien? Maar de nadruk ligt expliciet op het simpelst mogelijke ontwerp. Door middel van herontwerpen en refactoring kan de software altijd nog in een later stadium worden aangepast.

A.3.4 Test-Driven Development

Test-driven of test-first development (TDD) is binnen XP een hoeksteen van het eigenlijke programmeerwerk. Voor iedere User Story worden eerst (falende) geautomatiseerde tests geschreven. Pas daarna wordt de daadwerkelijke functionele code geïmplementeerd. Hiervoor identificeert Leffingwell drie hoofdredenen:

1. Door begrip van de tests wordt de ontwikkelaar gedwongen de User Story goed te begrijpen.
2. Een geautomatiseerde test die van tevoren wordt gemaakt, garandeert dat die nooit meer achteraf moet worden geschreven (in termen van technical debt).
3. De kwaliteit van geschreven code wordt hoger doordat alle code getest is.

TDD wordt dus gezien als een soort kwaliteitsgarantie voor de opgeleverde software: een baseline die hoger ligt dan wanneer slechts code wordt opgeleverd.

A.3.5 Continuous Integration

Op zeer regelmatige basis (minimaal dagelijks) wordt de ontwikkelde code geïntegreerd en getest. Door dit heel vaak te doen, ontstaat er een korte feedbackloop terug naar de ontwikkelaar van de code. Als de ontwikkelaar dan tests en code schrijft die problemen opleveren, dan wordt hij daar heel snel van op de hoogte gebracht.

A.3.6 Research spikes

Research Spikes of Spike Solutions zijn kleine programmaatjes die de ontwikkelaar schrijft om onderzoek te doen naar mogelijke oplossingen voor moeilijke technische of ontwerpproblemen. In plaats van de oplossing direct in een complex systeem te bouwen, bouwt de ontwikkelaar een op zichzelf staande applicatie die het gegeven probleem moet oplossen.

Vaak worden Spikes gebruikt om het risico van technische problemen beter te kunnen inschatten. Ook kunnen Spikes helpen bij het beter inschatten van de kosten van een User Story. Het is een middel om in te zetten als deze zaken zich in de loop van een ontwikkeltraject voordoen.

A.3.7 Conclusie

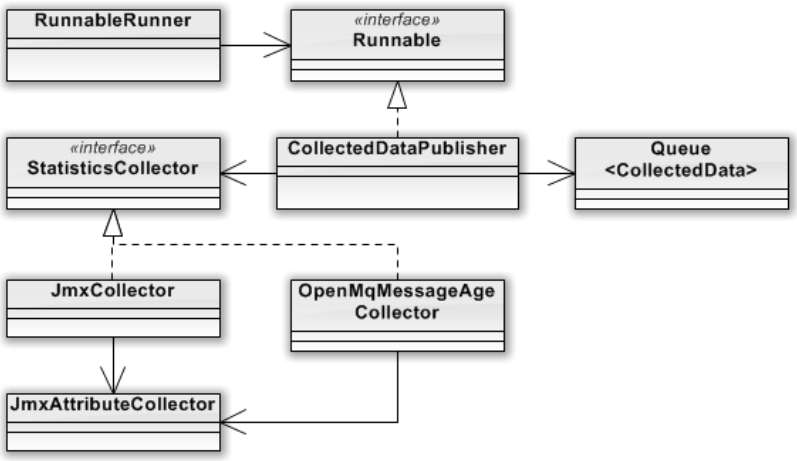
Door een aantal handige trucs en best practices over te nemen uit de Extreme Programming hoek, is het mogelijk om een agile invulling te geven aan de gaten die het Scrum model openlaat. Met deze methoden wordt het ontwikkelteam de mogelijkheid gegeven om effectief en efficiënt een product op te leveren van hoge kwaliteit.

Appendix B Technisch ontwerp

De monitor daemon bevat een aantal deelsystemen, elk met hun eigen verantwoordelijkheid. In dit appendix beschrijven we de werking hiervan in detail. De onderwerpen komen in dezelfde volgorde aan bod als in hoofdstuk 4.

B.1 Data-verzameling

Het verzamelen van data door Statistics Collectors wordt gestart door CollectedDataPublishers. Voor elke benodigde StatisticsCollector bestaat een instantie van CollectedDataPublisher. Bij de initialisatie wordt elke publisher als Runnable geregistreerd in een RunnableRunner. Deze wordt met een bepaalde frequentie uitgevoerd, waarbij elke CollectedDataPublisher een run()-aanroep krijgt. Excepties worden afgevangen, maar de betreffende publisher wordt niet nog een keer gestart. De standaardfrequentie is 10 seconden, dit is in te stellen in monitor-daemon.properties (zie configuratie).



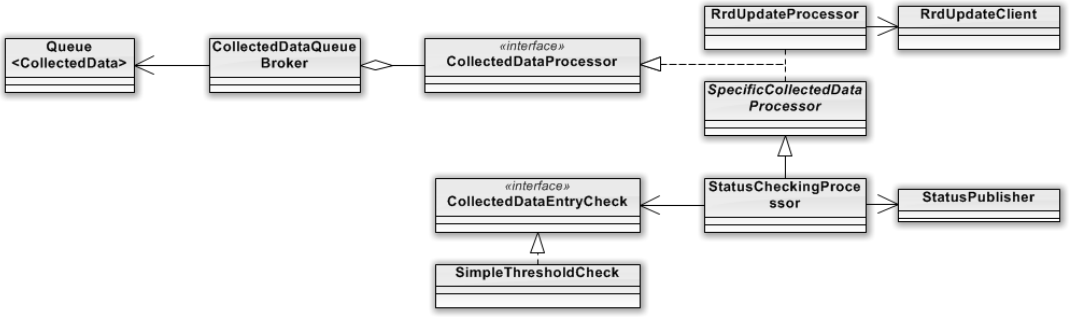
Figuur 17 Klassendiagram van data-verzameling

Er zijn twee belangrijke Statistics Collectors: de JMX- en de leeftijds-collector. Hoewel de leeftijd ook uit een JMX-opvraag afgeleid wordt, is er voor gekozen de leeftijd in de Collector uit te rekenen in plaats van bij de data-verwerkingsstap. Dit omdat de bestaande CollectedData-klasse niet gebouwd was op het doorsturen van de ruwe data in String-vorm, maar alleen op Numbers.

Het daadwerkelijk ophalen van de JMX-data wordt uitbesteed aan een JmxAttributeCollector. Dit gebeurde in ems-statistics-collector met een AbstractJmxCollector als parent van de JmxCollector. Bij het schrijven van de leeftijds-collector, de OpenMqAgeCollector, is besloten dit te herschrijven. Het makkelijker onderhouden was hier de belangrijkste factor in het kiezen voor een delegate boven overerving.

B.2 Data-verwerking

Een dataobject in de data-queue wordt opgepikt door de CollectedDataQueueRunner die het object doorgeeft aan de broker, een CollectedDataQueueBroker, die de data doorstuurt naar alle daar geregistreerde processoren. Deze processoren implementeren de CollectedDataProcessor-interface, zie ook het klassendiagram in .



Figuur 18 Klassendiagram van data-verwerking

Implementaties van deze interface CollectedDataProcessor zijn er voor het versturen van updates naar de RRD-server, en voor het checken tegen drempelwaarden voor Nagios.

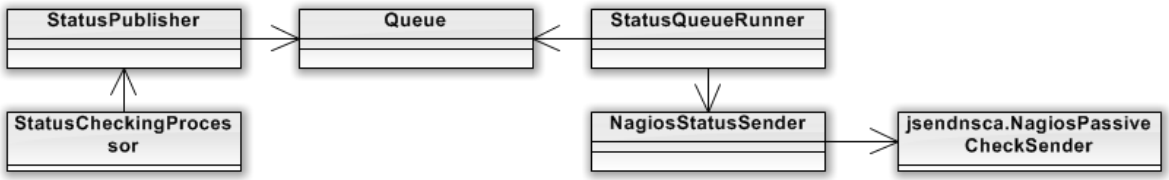
De RrdUpdateProcessor selecteert die CollectedData waarbij de rrdname ingesteld is, en stuurt ze aan de hand daarvan op naar de voorgeconfigureerde RRDTool-server.

Voor die CollectedData-name waarvoor een check geconfigureeerd is, is er een StatusCheckingProcessor. Dit is een extensie van de klasse SpecificCollectedDataProcessor, die de CollectedData op de juiste name filtert. Voor die CollectedData checkt de StatusCheckingProcessor of de goede Entry aanwezig is, en zo ja, geeft deze door aan zijn CollectedDataEntryCheck.

De implementatie daarvan is de SimpleThresholdCheck, die de Entry-waarde met een WARNING en een CRITICAL-drempel vergelijkt en een bijbehorend CheckResult teruggeeft. De StatusCheckingProcessor maakt hier een Status van door de voor Nagios benodigde Service name toe te voegen, en stuurt deze naar Nagios met de StatusPublisher.

B.3 Statusupdate versturen

In deze paragraaf gaan we in meer detail in op het opsturen met behulp van de JSendNSCA-bibliotheek en het verwerken van de berichten op de Nagios server.



Figuur 19 Klassendiagram van het Status opsturen

Het resultaat van het checken van de gegevens in een dataobject is een Status voor in Nagios. Deze wordt in een status-queue geplaatst. Als er genoeg beschikbaar zijn, of als er een bepaalde tijd verstreken is, worden deze opgestuurd met de NagiosStatusSender. Zie ook het klassendiagram in Figuur 19. Deze zet bij het versturen het interne Status-formaat om in het MessagePayload-formaat van de JsendNSCA-bibliotheek.

Voor het versturen van meerdere berichten tegelijk is NagiosPassiveCheckSender uitgebreid met een functie *send(List<MessagePayload> list)*, als variant op de bestaande *send(MessagePayload*

payload). In de nieuwe functie worden de payloads als losse pakketjes over één verbinding opgestuurd, met behulp van de in JSendNSCA aanwezige “inpak”-methoden.

De NSCA-daemon op de Nagios server schrijft de binnenkomende data naar de external command file, een speciale file waaruit het Nagios-proces zijn checkresultaten inleest. De frequentie waarmee Nagios deze file inleest is in de Nagiosconfiguratie in te stellen, evenals de grootte van de aan te houden buffer voor de file.

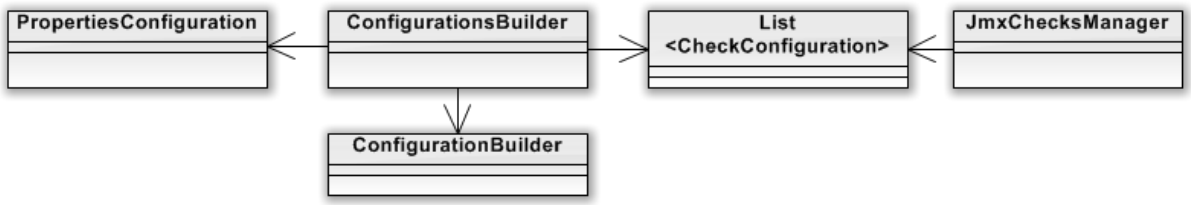
B.4 Configuratie

De configuratie van enomon vindt plaats in propertiesbestanden, tekstbestanden met key-value paren. De structuur van de applicatie is gecodeerd in XML-bestanden, die waarden uit de propertiesbestanden halen.

Algemene kernel-configuratie staat in properties-bestanden in de hoofdmap. De module configuratie-bestanden, die bestaan uit Spring context-xml-bestanden en properties, staan in submappen. Uit de submap met XML-bestanden worden alle aanwezige XML-bestanden geladen, zodat dit niet apart ingesteld hoeft te worden bij het installeren van nieuwe modules.

De standaard properties bestanden worden met Spring ingeladen, waarna de properties voor de checks met een PropertiesConfiguration van Apache Commons worden ingeladen, wat meer bewerkingsmogelijkheden geeft dan de standaard Properties klasse.

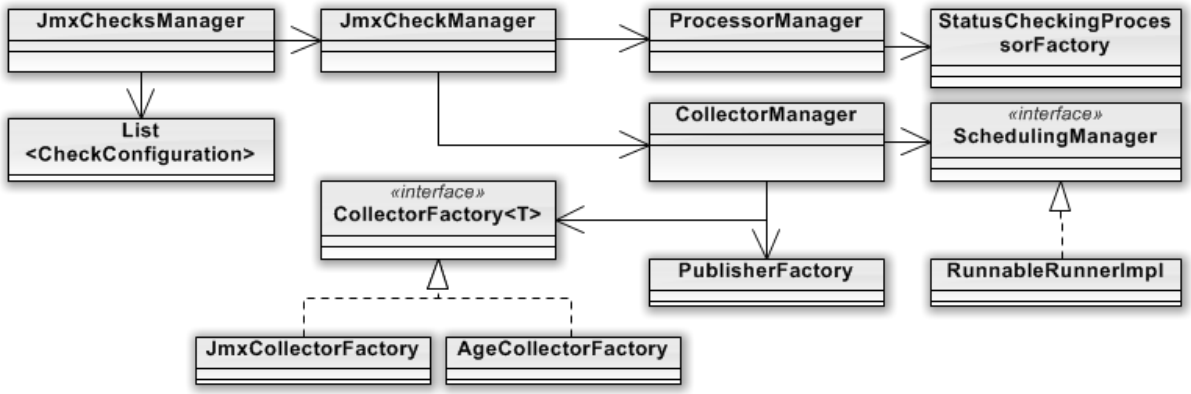
Gegeven deze PropertiesConfiguration, worden de losse checks hierin geïdentificeerd aan de hand van de sleutel. Deze dient te beginnen met een ingestelde string-prefix. Deze identificatie gebeurt in de Configurationsbuilder. Voor elke check wordt vervolgens een CheckConfiguration-object aangemaakt in de ConfigurationBuilder. In het CheckConfiguration-object worden alle benodigde en aanwezige configuratiewaarden opgeslagen.



Figuur 20 Klassendiagram van configuratieverwerking

B.5 Initialisatie

De zo geproduceerde lijst van CheckConfiguration instanties worden vervolgens door Manager en Factory-klassen gebruikt om de uitvoerende klassen te initialiseren en aan elkaar te hangen.



Figuur 21 Klassendiagram initialisatie

De JMX en de leeftijdchecks worden gedeeltelijk apart geinitialiseerd (eigen CheckConfiguration, eigen Managers), omdat ze op verschillende manieren data ophalen en weergeven. Het verwerken is op dit weergave-format na gelijk. In het klassendiagram in Figuur 21 zijn de initialiserende klassen voor de JmxCheck weergegeven, met de CollectorFactory als voorbeeld van de opsplitsing in Jmx en leeftijdsklassen.

Bij het initialiseren wordt door de CheckManager de verwerking in de ProcessorManager, en de verzameling in de CollectorManager gestart. In de CollectorManager wordt naast de statistics collector, ook de publisher aangemaakt, die in de uit te voeren publisherverzameling toegevoegd wordt.

De centrale configuratie met CheckConfiguration en Manager zorgt er zo voor dat de verzamelende en verwerkende klassen op de goede manier verbonden en opgestart worden.

B.6 Selfcheck

Los van de algemene initialisatie, wordt ook de selfcheck geinitialiseerd. Deze wordt samen met de andere checks uitgevoerd, en stuurt dan een ‘Status OK’-bericht naar Nagios, zodat daar te zien is dat het proces werkt, ook als de rest geen resultaat geeft, of er verder geen checks zijn ingesteld.

Appendix C Verklarende woordenlijst

- Daily Standup Meeting, DSM – De dagelijkse bijeenkomst van 15 minuten, waarin de stand van zaken in de Sprint wordt besproken.
- Hudson – Continuous integration server: http://www.hudson-ci.org/
- JMS – Java Message Service. Een standaard API voor de uitwisseling van berichten.
- JMX – Java Management Extensions. Een standaard API die gebruikt kan worden om verschillende soorten resources te benaderen. Een voorbeeld is de beschikbaarheid door middel van JMX van gegevens van de OpenMQ server.
- JSendNSCA – Een bibliotheek waarmee status updates kunnen worden gestuurd met behulp van het NSCA protocol naar de NSCA daemon: http://code.google.com/p/jsendnsca/
- Maven – Software project management tool: http://maven.apache.org/
- Nagios – Een IT infrastructuur monitoring tool: http://www.nagios.org/
- NSCA – Nagios Service Check Acceptor protocol. Een protocol waarmee status updates kunnen worden gestuurd, van een monitor agent op een te monitoren server, naar de centrale Nagios server.
- OpenMQ – Een implementatie van de JMS specificatie: http://openmq.dev.java.net/
- Product Owner – De vertegenwoordiger van de wensen van de klant.
- Quartz – Een uitgebreid job scheduling framework: http://quartz-scheduler.org/
- RPM – De RPM Package Manager, een tool waarmee packages kunnen worden geïnstalleerd, ook de afkorting die gebruikt wordt om RPM packages aan te duiden.
- Scrum Master – De procesmanager binnen het Scrum raamwerk.
- Scrum Team – Het ontwikkelteam, de Scrum Master en de Product Owner.
- Sonar – Een source code kwaliteit monitoring tool: http://sonarsource.org/
- Sprint – Een periode van twee tot vier weken waarin een nieuw “product increment” wordt ontwikkeld.
- Spring Framework – Een uitgebreid Enterprise Java ontwikkelframework: http://springframework.org/. Zie het Oriëntatieverslag in Appendix D voor meer informatie.
- Springsource Toolsuite, STS – Een op Eclipse gebaseerde ontwikkelomgeving die bedoeld is voor softwareontwikkeling met het Spring Framework.
- Subversion – Source Code Management server: http://subversion.apache.org/
- Team – Het team van developers binnen het Scrum Team.
- Wiki – Een softwarepakket dat het aanmaken van een verzameling van hyperlinked documenten eenvoudig maakt.
- XP – eXtreme Programming. Een agile ontwikkelmethode die gericht is op de programmeur en veel best practices beschrijft die ook in andere methoden kunnen worden toegepast: http://extremeprogramming.org/.
- XPlanner – Project planning en tracking tool voor XP teams. Primair een tool om het Product Backlog te managen: http://www.xplanner.org/

Appendix D Oriëntatieverslag

Oriëntatieverslag
Bachelor Project bij E.Novation

David Hartveld Piet-Jan Spaans

3 mei 2010

Voorwoord

Beste lezer,

Voor u ligt het verslag van de Oriëntatiefase van ons Bachelor Eindproject in het kader van onze opleiding Technische Informatica aan de Technische Universiteit Delft. Dit project voeren wij uit in de vorm van een stage bij de afdeling Application Development van E.Novation, IT-dienstverlener uit Capelle aan den IJssel.

Bij E.Novation werken we aan het verzelfstandigen en veralgemeniseren van de monitoring daemon en statistics collector van de E.Novation Message Service (EMS). Daarbij worden we begeleid door Bastiaan Bakker, Teamleider van het EMS ontwikkelteam. Vanuit de TU Delft wordt een oogje in het zeil gehouden door Peter Kluit, onze directe begeleider.

Dit verslag is tot stand gekomen na de eerste twee weken van onze stage. Hierna zal de implementatiefase gaan lopen gedurende acht weken. Aan het eind van de stage schrijven we het eindverslag.

David Hartveld & Piet-Jan Spaans

Inhoudsopgave

1 Introductie: Stagelopen bij E.Novation **5**
- 1.1 Applicatieontwikkeling bij E.Novation: het EMS team 5
- 1.2 Onze opdracht 5
- 1.3 Inhoud van het verslag 5

2 Beschrijving huidig systeem **6**
- 2.1 Globale architectuur 6
- 2.2 Het Spring framework 6
  - 2.2.1 Core en Context 6
  - 2.2.2 DAO en ORM 8
  - 2.2.3 JMS en OpenMQ 8
  - 2.2.4 JMX-interactie 8
  - 2.2.5 Overige componenten 9
- 2.3 Interactie tussen Nagios en de monitor daemon 9
  - 2.3.1 Aanroep door Nagios 9
  - 2.3.2 Uitvoering door monitoring daemon 9
  - 2.3.3 Terugsturen van het resultaat 9
- 2.4 Interactie tussen de statistics collector en RRDtool 9
  - 2.4.1 Statistiekverzameling 10
  - 2.4.2 RRD-update 10
- 2.5 Open source 10
- 2.6 Deployment 10

3 Project doelen: Wat gaan wij doen? **10**
- 3.1 De nieuwe daemon 11
  - 3.1.1 Optioneel: Dynamische architectuur 11

4 Mogelijke oplossingen **11**
- 4.1 Scheduling 11
  - 4.1.1 Implementatie van NMS 11
  - 4.1.2 Implementatie van agent 12
  - 4.1.3 Gekozen oplossing 12
- 4.2 Modulariteit 12
  - 4.2.1 Handmatig class-loading toepassen 13
  - 4.2.2 OSGi 13
  - 4.2.3 Java Plugin Framework 13
  - 4.2.4 Java Simple Plugin Framework 13

5 Sprint retrospective: Oriëntatiefase **13**
- 5.1 Gebruik story board 13
- 5.2 Formaat stories 14
- 5.3 Productiviteit 14

6 Inhoudelijke evaluatie oriëntatiefase **14**

A Nieuwe technologie: OSGi **15**
A.1 Bundles en hun lifecycle 15
A.2 Het service registry 15
A.3 OSGi R4.2 implementaties 15
A.4 Service component frameworks 16
A.4.1 Declarative services 16
A.4.2 Spring Dynamic Modules: de Blueprint Container RI . . 17
A.4.3 Apache Felix iPOJO 17
A.4.4 Peaberry met Google Guice 17

Lijst van figuren

1 Data Flow Diagram voor het huidige systeem 7
2 OSGi Bundle Life-Cycle 16

1 Introductie: Stagelopen bij E.Novation

E.Novation is een automatiseringsbedrijf dat IT-diensten aan andere bedrijven levert. Een van die diensten is de E.Novation Messaging Service (EMS), een secure messaging service voor onder andere de zorgsector (ZorgMail) en Rijkswaterstaat (Binnenvaart Informatie- en Communicatiesysteem BICS). Sinds de jaren '80 ontwikkelt E.Novation de berichtenservice en is daarin marktleider geworden. E.Novation is gevestigd in Capelle aan den IJssel op bedrijventerrein Rivium.

1.1 Applicatieontwikkeling bij E.Novation: het EMS team

Binnen E.Novation zijn wij actief bij de afdeling Application Development. Daar zijn twee ontwikkelteams. Het ontwikkelteam dat aan de EMS werkt, wordt geleid door Bastiaan Bakker, onze stagebegeleider. De afdeling werkt met een agile software ontwikkelingsproces. Wij zullen tijdens de hele stage ook volgens dit proces werken. Dit hebben we uitgebreid beschreven in het plan van aanpak.

1.2 Onze opdracht

De EMS bevat een monitoring daemon en een statistics collector. De monitoring daemon voert tests uit op de beschikbaarheid en kwaliteit van de service. De statistics collector houdt de hoeveelheid doorgevoerde berichten en de serverbelasting bij, waarna dit in grafieken zichtbaar gemaakt kan worden. Beide componenten zijn essentieel bij het in gebruik hebben van de service en worden gebruikt om te controleren of de service goed werkt.

Onze opdracht bestaat uit het losmaken en samenvoegen van deze twee componenten uit de EMS. De componenten zullen een zelfstandige service gaan vormen, die in de toekomst ook bij andere systemen ingezet kan worden.

Door de componenten te combineren wordt bovendien overlap tussen de componenten verwijderd. Verder moet de functionaliteit van de twee componenten gekruist worden: er kan gemonitord gaan worden op statistiek, en er kan statistiek worden bijgehouden van service monitoring.

1.3 Inhoud van het verslag

In de eerste twee weken van onze stage hebben we ons georriënteerd op de opdracht en de context van E.Novation waarin we zullen gaan werken. Hiervan doen we verslag in dit rapport. Alleen het ontwikkelproces is niet in dit verslag beschreven, maar staat in het Plan van Aanpak.

In hoofdstuk 2 beschrijven we de huidige opzet van het EMS, inclusief monitor daemon en statistics collector, en de interactie tussen de verschillende componenten. Vervolgens hebben we in hoofdstuk 3 de doelen van onze stage uiteengezet. Daarna beschrijven we een aantal mogelijke oplossingen in 4. Het verslag sluiten we af met een sprint retrospective in hoofdstuk 5. Hier evalueren we het verloop van de eerste (oriëntatie)sprint. In appendix A beschrijven we het OSGi Service Platform, een mogelijke oplossing voor het modulariteitsprobleem. Hoewel we dit framework voorlopig niet gebruiken, hebben we de beschrijving voor de volledigheid als bijlage toegevoegd.

2 Beschrijving huidig systeem

Om een goede nieuwe service te bouwen, is het belangrijk te weten wat de opzet is van het huidige systeem. We beschrijven de globale architectuur van het EMS-systeem, het Spring Framework waar het op gebouwd is, de interactie tussen het systeem en Nagios en RRDtool, en het uitrollen van componenten van de EMS.

2.1 Globale architectuur

De E.Novation Messaging Service is in staat om berichten van verschillende bronnen te verwerken, op te slaan en door te sturen. Dit is verdeeld over een aantal servers, met een loadbalancer en een monitoring server. In de ontwikkelomgeving is dit gereduceerd tot twee servers en een monitoring server.

Op de hoofdserver draaien verschillende java-processen. Voor het begrip volgen we een e-mail door het systeem. E-mail komt binnen via SMTP in het sendmail-proces, die het aflevert bij de ems-lmtpd (EMS local mail transport protocol daemon). Deze archiveert de e-mail, en plaatst hem in de Archived-Queue die met OpenMQ geïmplementeerd is. Het ems-analyze-message proces haalt hem hier weer uit, analyseert het bericht en plaatst het in de Scanned-Queue. Via deze queue gaat het bericht dan naar de ems-message-router. Indien nodig stuurt de router het bericht naar een converter om het technisch formaat van het bericht aan te passen. Vervolgens komt het bericht terug bij de router. Die plaatst het bericht in een inbox, of stuurt het door naar een ander netwerk.

Er is een aparte server voor authenticatie, CAS, die gebruik maakt van de identiteitsserver idm. Deze authenticatie wordt onder andere gebruikt voor de webmailinterface(ems-mailwebservice) en de beheerinterface(ems-mcenter).

Zoals te zien in het dataflowdiagram in Figuur 1, wordt de monitoring daemon door Nagios gepolld om checks uit te voeren. Dit kan het testen van de beschikbaarheid van een proces zijn, maar ook het bewaken van queue-waardes of complexere dingen als de doorlooptijd van een email.

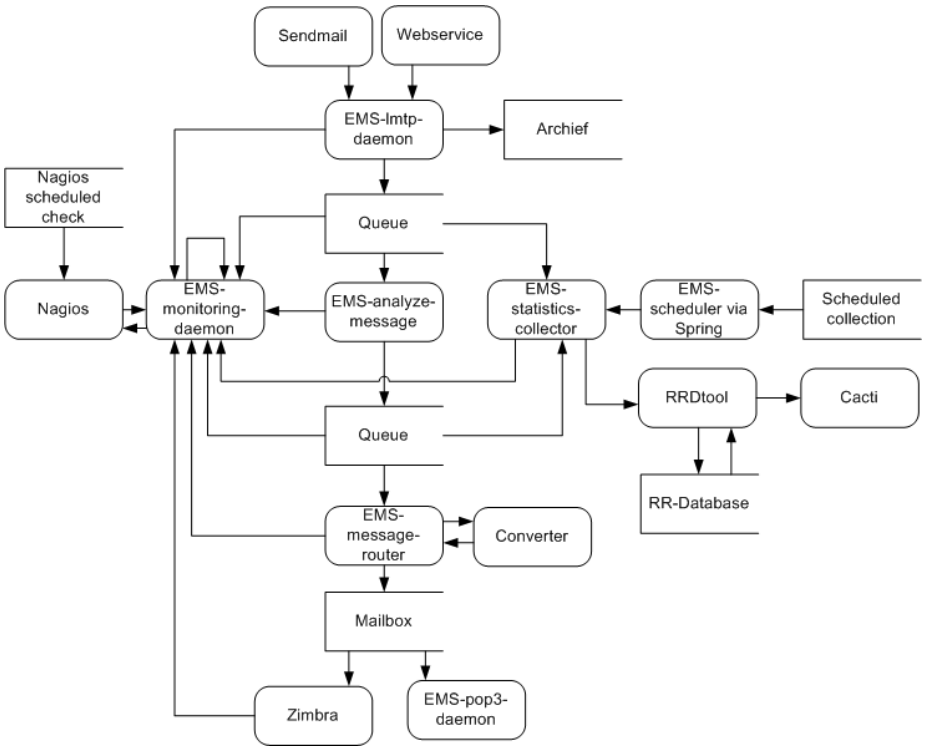
De statistics collector houdt zich bezig met het op regelmatige basis opslaan van bepaalde waarden, zoals het aantal threads of het aantal emails in een queue, en stuurt dit op naar RRDtool, waar het in een round-robin database komt (Een database die de gegevens maar een bepaalde tijd onthoud).

2.2 Het Spring framework

Alle EMS-componenten zijn gebouwd op basis van het Spring Framework. Het Spring Framework bestaat uit zes componenten: de Core, een Context-, DAO-, ORM-, AOP- en een Web-component. Delen hiervan worden door E.Novation gebruikt, en de Core is de belangrijkste functionaliteit waar wij mee te maken krijgen.

2.2.1 Core en Context

De Core en Context-componenten verzorgen het *Inversion of Control* ofwel *Dependency Injection* gedeelte. In XML-configuratiebestanden worden de verschillende classes als beans gedefinieerd. Voor elke bean kan worden aangegeven



Figuur 1: Data Flow Diagram voor het huidige systeem

welke constructor en setter-dependencies die heeft: instanties van andere beans die de bean nodig heeft om te kunnen functioneren. Spring zorgt er dan voor dat elke bean de classes waar hij van afhankelijk is, volledig geconfigureerd aangeleverd krijgt.

De specifieke toegevoegde waarde van de Context ligt in de extra functies die de klasse **ApplicationContext** biedt, waaronder **ApplicationEvent**s, een **MessageSource** voor applicatieberichten, makkelijkere bestands- en URL-toegang en de mogelijkheid meerdere contexts hiërarchisch te laden. Ook de monitoring daemon en statistics collector werken nu op basis van Dependency-Injected beans, en gebruiken de hiërarchisch inlaadmogelijkheid van de ApplicationContext.

2.2.2 DAO en ORM

Spring verzorgt ook een Data Acces Object laag, vooral om het benaderen van een database met directe JDBC onnodig te maken. Daarnaast ondersteunt Spring verschillende Object-Relational Mappings, waaronder Hibernate. Het EMS gebruikt Hibernate als Oject-Relational Mapping. Aangezien de monitoring daemon en de statistics collector beiden geen database nodig hebben, is dit voor ons minder interessant.

2.2.3 JMS en OpenMQ

De Java Message Service is een API voor het sturen van berichten. Het is de standaard die java enterprise applicaties in staat stelt om berichten te creëren, verzenden, ontvangen en lezen.

Open Message Queue (OpenMQ) is een open source JMS implementatie gemaakt door Sun. E.Novation gebruikt OpenMQ als messaging middleware. OpenMQ biedt het schaalbaarheid, een hoge beschikbaarheid, en een JMX beheer-API.

Spring voorziet in een interface over JMS heen, zodat er voor de gebruiker geen verschil is tussen de verschillende JMS-versies, en deze gebruik kan maken van de Spring templates en message listener containers.

In het huidige EMS-systeem wordt OpenMQ door de monitoring daemon en statistics collector benaderd via JMX.

2.2.4 JMX-interactie

Java Management Extensions is een API die systeemobjecten en apparaten van directe java-code scheidt door het gebruik van MBeans (Managed Beans), die een onderliggende resource vertegenwoordigen en beheren.

In het Spring framework is er de mogelijkheid elke Spring bean te gebruiken als JMX MBean, via een MBeanExporter. Dit kan door de bean te annoteren in de code, of door de bean expliciet mee te geven aan de exporter als MBean.

In de monitoring daemon en statistics collector worden MBeans benaderd om gegevens over bepaalde resources te verzamelen, maar worden geen MBeans aangeboden.

2.2.5 Overige componenten

Het Aspect-Oriented Programming gedeelte in Spring is er vooral om Aspects goed samen te laten werken met de Dependency Injection Core. Voor een volledigere Aspect-omgeving wordt door Spring zelf JAspect aangeraden. Aspect Oriented Programming is iets wat maar beperkt gebruikt wordt in het EMS, en niet in de componenten waar wij mee te maken hebben. We hebben daarom niet uitgezocht wat de mogelijkheden van deze module zijn.

De Web-component wordt op het moment in de monitoring daemon gebruikt om de webservice van het archief te benaderen. Dit kunnen we blijven gebruiken.

2.3 Interactie tussen Nagios en de monitor daemon

Deze paragraaf beschrijft hoe de Nagios server en de monitoring daemon in het huidige systeem samenwerken om de checks uit te voeren die E.Novation nodig heeft voor haar systeembeheer. In het huidige systeem zijn ongeveer 1700 checks geconfigureerd, waarvan een deel door de monitoring daemon wordt uitgevoerd.

2.3.1 Aanroep door Nagios

De Nagios server roostert op dit moment de checks in, die met Nagios Remote Plugin Execution (NRPE) worden uitgevoerd. Voor elke check wordt dan een proces gestart, dat via het netwerk contact maakt met de monitoring daemon. Doordat dit voor elke check een proces oplevert, zit de Nagios server in deze situatie snel aan zijn maximale procescapaciteit.

2.3.2 Uitvoering door monitoring daemon

Voor elke connectie vanuit Nagios krijgt de monitoring daemon via Apache Mina een thread met aanroep van MessageReceived in de MonitorHandler. Deze controleert de inkomende string op een correcte NRPE-syntax, zoekt deze op in een Map, en voert de bijbehorende check uit. Hiervoor wordt elke check expliciet in een command pattern gegoten: Elk CheckCommand heeft een check()-functie die door de opzoekfunctie kan worden aangeroepen.

2.3.3 Terugsturen van het resultaat

Na uitvoering van de check stuurt de monitoring daemon de status van de check als String terug naar Nagios. Dit bericht bevat een getal dat de status aangeeft, waarbij 0 OK is, 1 een waarschuwing en 2 een kritieke fout. Het Nagios checkproces schrijft dit resultaat in een bestand op de harde schijf. Nagios kan dit dan weer uitlezen en in een webinterface weergeven.

2.4 Interactie tussen de statistics collector en RRDtool

De statistics collector verzamelt statistieken over het EMS. Dit omvat onder andere de mailqueues en systeemeigenschappen als de gebruikte hoeveelheid geheugen en aantal threads. Deze data wordt gepusht naar de RRDtool-server.

2.4.1 Statistiekverzameling

Door de Quartz-module wordt in Spring van een lijst Collectors op gezette tijden de collect()-methode aangeroepen. Hierbij wordt met bijvoorbeeld een MBean of door directe JVM-aanroepen een waarde verzameld, die aan de data- Queue wordt toegevoegd.

2.4.2 RRD-update

Er is een thread voor de **SimpleQueueRunner**, die de data-Queue in de gaten houdt, zodat elk stukje nieuwe data los opgestuurd kan worden. Dit wordt door de **RrdUpdateClient** als string opgestuurd naar de RRDtool-server, die het verwerkt en hier de grafieken van maakt die in Cacti weergegeven kunnen worden.

2.5 Open source

Binnen de ontwikkelafdeling van E.Novation wordt de voorkeur gegeven aan het gebruik van open source software. Dit is voornamelijk zodat er bij problemen "binnenshuis"gekeken kan worden of de fout in de eigen, of in de externe software optreed. Bovendien kan dan ook intern het probleem opgelost worden, zonder dat op de leverancier hoeft te worden gewacht. Tot slot is het gebruik van open source pakketten vaak goedkoper. Deze voorkeur uit zich bijvoorbeeld in het gebruik van het Spring-framework, maar ook in CentOS op servers, Nagios als NMS, enzovoort.

2.6 Deployment

De daadwerkelijke uitrol van softwarepakketten vindt plaats door middel van RPMs. Deze worden bij iedere software build ook aangemaakt. Omdat builds per uitontwikkelde story worden gedaan, zijn RPMs dus ook aan stories gekoppeld.

3 Project doelen: Wat gaan wij doen?

De huidige monitoring daemon en statistics collector moeten ook voor andere systemen makkelijk inzetbaar worden, en moeten daarom los komen te staan van het EMS.

Daarnaast moet de Nagios server, die de status van onder andere de EMS controleert, passief status updates gaan verwerken in plaats van actief. Hiervoor moet een nieuwe, zelfstandige monitor daemon worden ontworpen, die de status updates gaat sturen.

Deze daemon gaat ook statistieken verzamelen en naar RRDtool sturen. Dit vervangt de functionaliteit van de huidige statistics collector binnen de EMS.

De monitor daemon moet op specifieke momenten of met een regelmatig herhalingspatroon bepaalde controles uitvoeren. Sommige controles moeten, voor zover dat kan, zo vaak mogelijk worden uitgevoerd. De status van die controles wordt vervolgens naar Nagios gestuurd.

Tot slot kan ook op basis van de verzamelde statistieken worden gekeken naar de status van een dienst. De statistieken worden dan tegen van tevoren

ingestelde predicaten getest. Het resultaat van die tests wordt vervolgens naar Nagios gestuurd.

3.1 De nieuwe daemon

De nieuwe daemon moet een flexibele architectuur hebben, met een algemene kern en optionele modules voor de specifieke systemen. Als deze checks dynamisch ingeladen kunnen worden, dan zou dat mooi zijn, maar dat is geen vereiste.

De monitor daemon moet de functionaliteit van de huidige EMS monitor daemon vervangen. Alle checks die nu uitgevoerd worden, moeten zo mogelijk gegeneraliseerd worden, en modulair deployable worden. Hetzelfde geldt voor de statistics collector: alle statistieken moeten ook door de nieuwe daemon worden verzameld. Verder moet de monitor checks op verzamelde statistieken gaan ondersteunen, en moeten statistieken verzameld kunnen worden van monitor checks.

3.1.1 Optioneel: Dynamische architectuur

De nieuwe daemon moet optioneel een flexibele architectuur krijgen die het mogelijk maakt om dynamisch nieuwe modules met monitor of statistiek checks te laden. Bij het laden van een module worden dan de checks geregistreerd in de daemon, inclusief informatie over hoe de checks moeten worden gescheduled. De scheduler weet zo hoe de geregistreerde checks moeten worden uitgevoerd. Als een module verwijderd wordt, worden de registraties behorende bij de checks in de module ook verwijderd.

4 Mogelijke oplossingen

Er zijn twee aspecten waar een goede oplossing voor moet komen: het schedulen van checks en de mogelijke dynamische architectuur van de nieuwe daemon.

4.1 Scheduling

De belangrijkste beslissing bij het inroosteren van de serverchecks is waar ze gescheduled worden: centraal, in het Network Management System, of decentraal, op de verschillende servers.

4.1.1 Implementatie van NMS

Voor het centraal uitvoeren van het schedulen van monitoring taken bestaan veel kant- en klare NMS-oplossingen. Nagios zelf zou dit ook kunnen, ware het niet dat het voor iedere check een systeemproces nodig heeft, wat relatief duur is.

Twee andere grote NMS-en zijn OpenNMS en Hyperic, beide op Java gebaseerd. Hierdoor kost een taak, ook al wordt er op het antwoord gewacht, slechts een java thread, wat goedkoper is dan een proces.

OpenNMS OpenNMS scheduled zijn eigen taken, en combineert hierbij netjes monitoring en statistiek. Via JMX of SNMP is de monitoring daemon te benaderen, waarna het resultaat door OpenNMS doorgestuurd wordt naar Nagios. De integratie van OpenNMS in Nagios wordt niet direct ondersteund, dus het doorsturen naar Nagios geeft dezelfde problemen als bij het zelf implementeren van de scheduling.

Hyperic HQ Hyperic bestaat uit een klein Hyperic HQ deel voor de monitoring server, en agents voor op de te monitoren servers. De HQ server pollt de agents wanneer dat nodig is, of als het gescheduled is.

4.1.2 Implementatie van agent

Voor het decentraal uitvoeren van de scheduling is een eigen implementatie de meest voor de hand liggende oplossing. Hierbij wordt de bestaande scheduling van statistiekverzameling zo gelaten, en wordt er voor de andere checks een eigen scheduling geschreven. Deze checks moeten zo vaak mogelijk uitgevoerd worden. Er zijn echter kort en lang durende checks. Een mogelijke, simpele oplossing is deze beide een eigen thread pool te geven.

Communicatie met NMS De agent heeft dan intern de resultaten van een check, en moet deze nog naar Nagios en RRDtool krijgen. Voor RRDtool kan de bestaande interface hergebruikt worden. Voor Nagios zijn er twee opties: het schrijven van een NSCA-client (voor het Nagios Service Check Acceptor protocol) of het schrijven van een SNMP-client, die SNMP Traps naar Nagios stuurt. Voor beiden zijn java libraries beschikbaar, en een serverdaemon om binnenkomende checks in de Nagios *External Command File* te schrijven. Voor elke binnenkomende check moet al een service in Nagios gedefinieerd zijn.

NSCA client Het voordeel van het gebruiken van de meegeleverde Nagios-technologie is dat het aan de serverkant gewoon werkt. Het nadeel is dat je voorlopig aan Nagios als NMS vast blijft zitten. De Nagios NSCA-client leest commands vanaf de standaard input. De Java-libraries NagiosAppender en JSendNSCA lijken een netter alternatief voor een Java-applicatie.

SNMP traps Door SNMP Traps te gebruiken, wordt de agent onafhankelijk van de NMS-implementatie. Nagios ondersteunt SNMP Traps zelf niet, maar er is het losse 'SNMP Trap Translator'- project dat de Traps kan opvangen en aan Nagios doorgeeft.

4.1.3 Gekozen oplossing

E.Novation heeft aangegeven de voorkeur te geven aan de implementatie van een agent, onder andere omdat decentrale scheduling makkelijker te implementeren.

4.2 Modulariteit

De wens om monitoring checks en statistiek collectors dynamisch als modules te kunnen laden in de nieuwe agent kan op verschillende manieren worden vervuld. Iedere methode heeft zo zijn voor- en nadelen. Die worden hierna besproken.

Hier is nog geen keuze in gemaakt, ook omdat het geen bijzonder dringende wens vanuit E.Novation is.

4.2.1 Handmatig class-loading toepassen

Het is mogelijk om een simpel plugin framework te bouwen, wat gebruikt kan worden om de gewenste modulariteit te bereiken. Bij het willen werken met verschillende versies van dezelfde class wordt dit al snel lastig om goed en op een nette manier te doen.

4.2.2 OSGi

We hebben in Appendix A uitgebreid beschreven wat de mogelijkheden zijn van het OSGi Service Platform. Een van de grote voordelen is de flexibiliteit van het platform, maar dit komt met een prijs: de complexiteit neemt hierdoor toe.

4.2.3 Java Plugin Framework

Het Java Plugin Framework, een wat ouder framework is gebaseerd op het oorspronkelijke plugin framework van Eclipse (v2 en eerder, vanaf v3 wordt OSGi gebruikt). Dit is geschikt om nieuwe klassen in te laden uit jar files. We weten op dit moment echter niet precies hoe de integratie met het Spring framework (of een ander dependency injection mechanisme) eruit moet gaan zien.

4.2.4 Java Simple Plugin Framework

Het Java Simple Plugin Framework is een wat moderner framework voor het eenvoudig laden van plugins. Ook voor dit framework geldt dat we geen duidelijk beeld hebben van hoe dependency injection zou kunnen werken.

5 Sprint retrospective: Oriëntatiefase

Aan het eind van de oriëntatiefase hebben we een sprint retrospective gehouden, volgens het model dat het EMS team aanhoudt. We hebben een aantal positieve en negatieve punten geïdentificeerd. De belangrijkste drie punten bespreken we hieronder. Dit zijn de punten waar we de komende sprint extra op zullen gaan letten.

5.1 Gebruik story board

We hebben het story board tijdens de eerste stageweek goed gebruikt. De stories waar we mee bezig waren, hingen in de kolom DOING, de overige in ofwel TODO, ofwel in DONE. We hielden hierdoor een duidelijk overzicht van waar we mee bezig waren, wat er nog moest gebeuren en waar we op moesten letten.

In de tweede week gingen we chaotischer aan de slag. Er hingen te veel stories in de kolom DOING, waardoor niet meer duidelijk was waaraan daadwerkelijk werd gewerkt. Daardoor raakten we het overzicht kwijt.

In de komende sprint gaan we hier extra op letten: op ieder moment kun je maar met één ding tegelijk bezig zijn, en dat moet gereflecteerd worden op het story board. Hierdoor blijft het overzicht expliciet bewaard, en dat is belangrijk in een Scrum ontwikkelproces.

5.2 Formaat stories

De reden dat we aan meerdere stories tegelijk bezig waren, was dat sommige stories te groot waren, en deels afhankelijk waren van andere stories. Hierdoor moesten we af en toe het werk aan grote stories onderbreken om aan andere stories te werken. Dit heeft overigens ook te maken met het feit dat we hebben geprobeerd een onderzoeksverslag te schrijven op basis van het kanban systeem, wat niet erg goed matcht.

5.3 Productiviteit

Onze productiviteit is de afgelopen twee weken wisselend geweest. Dit heeft twee oorzaken gehad.

De eerste is dat we soms te lang met een taak bezig bleven, terwijl er maar een beperkte hoeveelheid tijd voor beschikbaar was. Dit waren vaak uitzoek- of onderzoekstaken, waar we dan te diep op in gingen. Het resultaat is dat we sommige onderwerpen ver hebben uitgezocht (bijvoorbeeld de werking en mogelijkheden van het OSGi Service Platform), die we mogelijk helemaal niet gaan gebruiken. Bovendien hebben we andere zaken niet ver genoeg uitgezocht. Dat moet nu in de implementatiefase gaan gebeuren en kost dan extra tijd.

De tweede oorzaak van vertragingen was dat we soms te lang op feedback moesten wachten als we die nodig hadden. We hebben de verschillende betrokkenen te laat ingelicht dat we feedback nodig hadden, waardoor ze die niet op tijd konden geven. Dit heeft ervoor gezorgd dat we soms niet even effectief konden opereren. Door een betere planning te maken, en daarover beter te communiceren, verwachten we dat we dit probleem wel op kunnen lossen.

6 Inhoudelijke evaluatie oriëntatiefase

Aan het eind van de oriëntatiefase kwamen we tot de conclusie dat we een aantal onderwerpen onderzocht hadden, die eigenlijk niet zo belangrijk waren. Het grootste voorbeeld daarvan is ons onderzoek naar het OSGi platform. Pas in het laatste gesprek met Bastiaan Bakker op donderdag kwamen we tot de conclusie dat gebruik van OSGi eigenlijk helemaal niet nodig is voor de implementatie van onze opdracht. Aangezien we veel tijd hebben besteed aan het uitzoeken van de mogelijkheden van dit platform, kun je de vraag stellen of we onze tijd niet nuttiger hadden kunnen besteden. Hetzelfde geldt een beetje voor ons onderzoek naar Hyperic HQ en OpenNMS, hoewel we daar wel onze kennis van het huidige systeem mee hebben kunnen verbeteren.

In plaats daarvan hadden we een veel praktischer benadering moeten nemen. Hoe werkt het huidige systeem nou eigenlijk en wat kunnen we daaraan aanpassen, zonder dat er enorm veel moet veranderen aan de opzet van het systeem? Wat is urgent en belangrijk dat gaat gebeuren? Wat is minder urgent? Welke ideeën heeft E.Novation daarover?

Gelukkig hebben we dit in hoofdlijnen wel duidelijk. Er is in samenspraak met Bastiaan een backlog opgesteld dat de weg wijst voor de komende sprint. Omdat stories heel kleine, precieze omschrijvingen zijn van wijzigingen in het systeem, kunnen we daar niet zo makkelijk de verkeerde weg mee inslaan. Het is wel van essentieel belang dat onze communicatie met Bastiaan op regelmatiger basis plaats gaat vinden en dat hij goed op de hoogte blijft van waar wij mee

bezig zijn. Dat moet met zijn aanwezigheid bij de daily standup meetings in orde komen.

A Nieuwe technologie: OSGi

Het dynamisch in kunnen laden van nieuwe monitoring checks is een van de wensen voor de nieuwe daemon. On-the-fly moeten checks kunnen worden toegevoegd en geüpdate. In de Java wereld is er eigenlijk maar een raamwerk dat hiervoor geschikt is: het OSGi Service Platform. [1]

Dit platform biedt een omgeving waarin deployment van applicaties flexibel kan worden uitgevoerd: de dependencies en de applicatie zelf kunnen flexibel worden aangepast en geüpdate, zonder dat de werking van de applicatie in gevaar komt.

A.1 Bundles en hun lifecycle

Het OSGi platform is een runtime die het on-the-fly laden van bundels, reguliere jar-files met een aantal extra manifest headers, mogelijk maakt. Hierdoor kunnen de klassen die de bundel bevat beschikbaar worden gemaakt aan de reeds draaiende applicatie.

De lifecycle van een bundel is gedefinieerd op zo'n manier dat het (bundelinterne) management eenvoudig is. De OSGi runtime draagt zorg voor alle states en transities in figuur 2, behalve de STARTING en STOPPING state. De bundel kan zichzelf configureren in die laatste twee states. Bij het laden van een bundel controleert de runtime of er een klasse geladen is die de interface **BundleActivator** implementeert. Als dat zo is, dan wordt een instantie van die klasse aangemaakt. Vervolgens worden bij respectievelijk het starten en stoppen van de bundel de in BundleActivator gedefinieerde methoden start(...) en stop(...) aangeroepen.

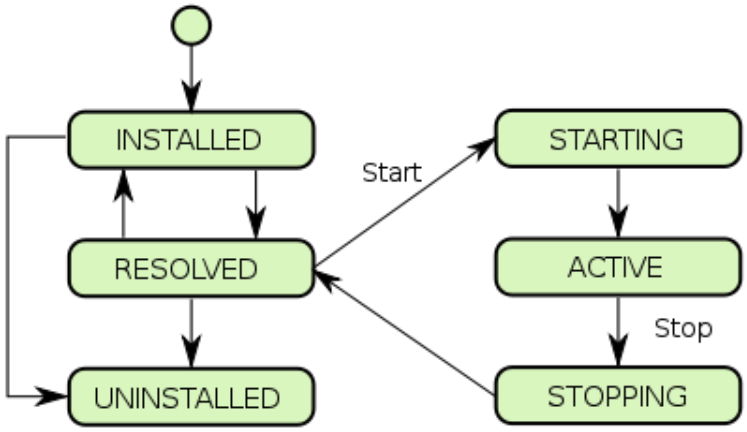
A.2 Het service registry

De interactie tussen bundels vindt plaats via services. Dit zijn (Java) interfaces die geïmplementeerd worden door klassen in bundels. Bundels kunnen services exporteren door implementerende objecten te registreren in het service registry, een centraal register in de runtime. Dit kan bijvoorbeeld gebeuren in de STARTING state. In de STOPPING state kan de service registratie vervolgens worden verwijderd.

A.3 OSGi R4.2 implementaties

Er zijn verschillende open source implementaties van de OSGi specificatie (Release 4.2). De bekendste zijn Apache Felix, Knoplerfish en Eclipse Equinox (de basis van de Eclipse runtime, RCP en IDE). Deze producten implementeren de core OSGi specificatie allen volledig, wat betekent dat ze ook volledig compatibel zijn: OSGi bundles kunnen in elk van de drie zonder problemen gebruikt worden. Daarmee is er geen noodzaak om een keuze tussen deze implementaties als definitief te zien.

[1] OSGi Service Platform: http://www.osgi.org/About/WhatIsOSGi



Figuur 2: OSGi Bundle Life-Cycle

Een vergelijking van Apache Felix en Eclipse Equinox als containers: Apache Felix is een implementatie onder Apache licentie, met goede documentatie. Het wordt gebruikt door onder andere EasyBeans, NetBeans en GlassFish. Eclipse Equinox is de ruggengraat van Eclipse, en valt ook onder de Eclipse Public License. De documentatie is redelijk, maar iets minder toegankelijk dan die van Felix. Equinox wordt gebruikt in onder andere Eclipse en Spring DM Server.

A.4 Service component frameworks

Om het werken met OSGi services eenvoudiger te maken, kan gebruik worden gemaakt van service component frameworks. Deze frameworks vereenvoudigen het gebruik van services uit andere bundels en het exporteren van services aan andere bundels. De hoeveelheid boilerplate code die nodig is om services te importeren en exporteren wordt hierdoor drastisch gereduceerd. Het is zeer sterk aan te bevelen om een dergelijk framework te gebruiken.

Alle hieronder genoemde service component frameworks gebruiken het OSGi service registry als onderliggende basis. Hierdoor zijn de services op interbundel niveau volledig compatibel.

A.4.1 Declarative services

Het produceren en consumeren van services binnen het OSGi services framework was traditioneel relatief ingewikkeld. De hoeveelheid boilerplate code was erg groot. Dit probleem werd onderkend en hiervoor werd de Declarative Services (DS) specificatie samengesteld. De DS specificatie voorziet in een zeer basale

vorm van service injection. Services in de ene module worden met behulp van XML bestanden gemarkeerd als produced, en in een andere (consumerende) module als required. Het werk van het koppelen van referenties wordt vervolgens automatisch gedaan door de DS layer.

A.4.2 Spring Dynamic Modules: de Blueprint Container RI

De OSGi DS specificatie heeft een aantal beperkingen, en is nog steeds niet ideaal om mee te werken. Het Spring framework biedt een dependency injection container. Omdat integratie van services als injectable beans erg handig zou zijn, is Spring Dynamic Modules (DM) ontwikkeld. Dit framework is aangeslagen en op basis hiervan is een nieuwe specificatie voor OSGi ontwikkeld: De Blueprint Container specification. Spring Dynamic Modules is hiervan de reference implementation[2]. Het gebruik van Spring DM impliceert gebruik van het Spring framework: DI van gewone beans binnen een bundel behoort ook tot de geboden functionaliteit.

A.4.3 Apache Felix iPOJO

Apache Felix iPOJO is qua functionaliteit vergelijkbaar met Declarative Services, maar is veel geavanceerder. Er is vrijwel geen boilerplate code nodig. Performance is goed, onder andere door een systeem waarbij compile-time bytecode manipulation wordt gebruikt om service references te injecteren. Dependency injection van beans binnen de bundel ontbtreekt.

A.4.4 Peaberry met Google Guice

Peaberry is vergelijkbaar met Spring Dynamic Modules. Op basis van Google Guice wordt DI functionaliteit aangeboden, voor services uit externe bundels, en voor beans binnen de bundel. Het is echter geen implementatie van OSGi Blueprint Container.

[2]Spring DM: http://static.springsource.org/osgi/docs/2.0.0.M1/reference/html/why-Spring%20DM.html

Appendix E Plan van aanpak

Plan van Aanpak
Bachelor Project bij E.Novation

David Hartveld Piet-Jan Spaans

3 mei 2010

1 Introductie: Stagelopen bij E.Novation

Dit Plan van Aanpak beschrijft onze stage bij E.Novation vanuit het procesperspectief. Hoe werkt de afdeling Application Development, en hoe gaan wij daarin mee draaien?

1.1 Projectopdracht

Onze opdracht bestaat uit het losmaken en samenvoegen van de monitoring daemon en statistics collector uit de E.Novation Message Service. De componenten zullen een zelfstandige service gaan vormen, die in de toekomst ook bij andere systemen ingezet kan worden. De opdracht staat in detail beschreven in het oriëntatieverslag.

1.2 Inhoud van het Plan van Aanpak

Dit Plan van Aanpak beschrijft het ontwikkelproces zoals dat bij E.Novation wordt toegepast. Dit is in hoofdstuk 2 terug te vinden. Onder andere de Scrum methodologie wordt beschreven in paragraaf 2.1. De betrokken bij het proces worden beschreven in paragraaf 2.2 en de manier waarop sprints verlopen in paragraaf 2.3. In paragraaf 2.5 wordt het test- en QA-proces beschreven.

Het Plan van Aanpak wordt afgesloten met een overzicht van de te bereiken stagedoelen in hoofdstuk 3.

2 Het ontwikkelproces bij E.Novation

Op de afdeling Application Development werken twee kleine ontwikkelteams van vier personen. Verder zijn er een QA-team en een applicatiebeheerteam. Een van de twee teams is het EMS-team, met als team leader Bastiaan Bakker, onze stagebegeleider. Hij zal een mini-ontwikkelteam leiden, met ons als leden. Bij E.Novation wordt een agile softwareontwikkelingsproces gevolgd. In dit hoofdstuk beschrijven we hoe dit proces er uit ziet, en hoe wij het in onze stage zullen gaan toepassen.

2.1 Scrum

Het proces vindt plaats binnen het Scrum raamwerk. Daarbij wordt een softwareontwikkelingsproject ingedeeld in sprints: korte periodes van enkele weken, waarbij incrementeel aan het product wordt gewerkt, en waarbij na elke sprint steeds een verder uitgebreid, bruikbaar product wordt opgeleverd. Bij E.Novation wordt meestal met sprints van drie weken gewerkt. Tijdens onze stage zullen we vijf sprints van twee weken uitvoeren.

2.2 Betrokken teams en personen

Bij het ontwikkelen van software bij E.Novation zijn verschillende personen en teams betrokken. Het ontwikkelteam bestaat uit team leden en een scrum master. De scrum master heeft als taak ervoor te zorgen dat obstakels voor het proces worden opgeruimd. De team leden doen het eigenlijke ontwikkelwerk. Er is verder een product owner: Dit is de persoon die de belangen van de klant behartigt en de produktontwikkeling de goede kant op stuurt.

Zodra een nieuwe feature ontwikkeld is, wordt deze door het QA-team getest. Wanneer een applicatie in productie wordt genomen, wordt deze beheerd door het team Technisch Applicatiebeheer.

Tijdens onze stage zal Bastiaan Bakker product owner zijn. We zullen de taken van de scrum master om en om uitvoeren.

2.3 Sprints

Het doel van een sprint is het opleveren van een nieuwe, uitgebreide versie van een product. Die nieuwe versie moet toegevoegde waarde leveren aan de klant ten opzichte van het bestaande product. De te realiseren uitbreidingen worden vastgelegd in het product backlog, een geprioriteerde lijst met wensen die de klant gerealiseerd wil zien. Deze wensen worden in stories uitgedrukt. Dit zijn beschrijvingen van de kleinste stukjes functionaliteit, die toegevoegde waarde bieden aan de klant, als deze geïmplementeerd zijn. Hierdoor zijn de stories makkelijker en sneller te implementeren en vooral ook volledig af te ronden. Stories bestaan typisch wel uit meerdere taken die de ontwikkelaar moet uitvoeren om de story af te ronden: uitzoeken, programmeren, documentatie schrijven, etc. zijn allemaal taken die binnen een story kunnen voorkomen. Normaliter duurt het compleet afronden van een story meer dan een dag, maar meestal niet langer dan een week.

2.3.1 Planning van de sprint

Voordat een sprint kan worden gestart, moet er een geprioriteerde, met stories gevuld product backlog zijn. De product owner bepaalt welke stories in het product backlog terecht komen. Deze stories kunnen nog niet geïmplementeerde stories zijn uit een vorige sprint, of nieuwe stories die de product owner aandraagt.

Als de stories in het backlog zijn vastgesteld, zal het ontwikkelteam van iedere story vaststellen hoeveel story points die krijgt - een relatieve indicator van de tijd en moeite die het gaat kosten om de story te implementeren. Vervolgens zal de product owner de stories verschillende prioriteiten geven. De prioriteit

hangt af van de waarde die de stories toevoegen aan het eindproduct, ten opzichte van de hoeveelheid story points die voor die story staan. De stories zullen tijdens de sprint op volgorde van prioriteit worden geïmplementeerd. Hierdoor worden de belangrijkste, meest waardevolle stories als eerste geïmplementeerd, en wordt het resultaat van de sprint maximaal.

In principe wordt, voorafgaand aan de sprint, het backlog vastgesteld. Het is niet de bedoeling tijdens een sprint het backlog dan nog te veranderen. Normaliter worden nieuwe stories bewaard voor de volgende sprint. Toch komt het soms voor dat een klant per se een extra feature geïmplementeerd wil zien, of dat de klant tegen een kritieke bug aanloopt. Hiervoor worden dan nieuwe stories toegevoegd. Ook komt het voor dat het ontwikkelteam ontdekt dat er extra stories nodig zijn om een bestaande story te kunnen implementeren. Ook dan worden extra stories toegevoegd aan het backlog.

2.3.2 Voortgang binnen de sprint

Tijdens de sprint wordt dagelijks een korte bespreking gehouden van maximaal 30 minuten, waarbij voortgang van de sprint wordt besproken. Deze daily standup meeting wordt gehouden bij het story board - meestal een stuk muur, ingedeeld in een aantal kolommen: TODO, DOING, TO VERIFY, QA, DONE. De te implementeren stories uit het backlog van de sprint worden op story cards geprint en op de muur gehangen in de kolom TODO - nog te implementeren stories. Als een teamlid een story op zich neemt om uit te voeren, wordt de story verplaatst naar DOING - de stories die op dat moment geïmplementeerd worden. Ieder teamlid mag in de kolom DOING maximaal een story hebben hangen waarmee hij op dat moment bezig is. In de kolom TO VERIFY hangen de stories die geïmplementeerd zijn, maar die nog gecontroleerd moeten worden door een ander teamlid. Als een tweede teamlid de implementatie van de story gecontroleerd heeft, wordt de story in de QA kolom geplaatst. Dat betekent dat het QA-team acceptance tests voor de story gaat ontwikkelen en uitvoeren. Als de story daadwerkelijk als volledig en correct uitgevoerd wordt beoordeeld, wordt de story naar DONE verplaatst.

2.3.3 Afronding van de sprint

Aan het einde van een sprint wordt de nieuwe versie van het product opgeleverd, meestal in zo'n vorm dat de klant het product direct kan gebruiken. Soms wordt ook een presentatie gegeven van de gerealiseerde uitbreidingen ten opzichte van de vorige versie, niet alleen aan de klant, maar ook aan collega's binnen E.Novation.

De sprint wordt afgesloten met een evaluatie. Het eerste deel bestaat uit een standup meeting waarbij ieder teamlid drie positieve en drie negatieve punten over de afgelopen sprint aandraagt. Het gaat hierbij om punten die op het proces van de sprint van invloed zijn geweest. Vervolgens worden door ieder teamlid drie punten als belangrijk aangegeven.

Het team bespreekt daarna in een overleg de drie punten die door de meeste teamleden werden aangewezen. De conclusies die hieruit worden getrokken worden vaak kort en bondig op een A4tje gezet en bij het story board gehangen voor de volgende sprint.

2.4 Lean software development

De klant heeft niets aan het resultaat van een half geïmplementeerde story. Dit is de reden dat stories zo klein mogelijk worden gekozen: dan is de kans dat ze volledig afgerond worden maximaal. Binnen agile softwareontwikkeling is dit een essentieel gegeven. Alle tijd en energie die in een niet opgeleverde of benutte feature wordt gestoken, wordt beschouwd als trash - afval, nutteloos. Hieronder vallen dus onafgeronde stories, maar bijvoorbeeld ook features in het product die de klant eigenlijk helemaal niet nodig heeft. De nadruk ligt op het afleveren van een zo klein mogelijk product, dat zo goed mogelijk voldoet aan de wensen van die klant. Tijdens onze stage zal dit dus ook een belangrijke sturende factor zijn. Door incrementeel te ontwikkelen en steeds specifieke functionaliteit te implementeren, proberen we een functioneel en compleet eindproduct neer te zetten.

2.4.1 Deliverables

Tijdens iedere sprint wordt toegewerkt naar oplevering van een nieuwe versie van het te ontwikkelen product. De featureset die geïmplementeerd moet worden, wordt opgedeeld in verschillende stories. Iedere story heeft z'n eigen deliverables: source code, tests, packaging scripts, installatiehandleidingen, enz. Tegen het eind van de sprint worden de release notes voor de nieuwe release geprepareerd, die voornamelijk bestaan uit een installatie howto, en eventueel een beschrijving van de toegevoegde featureset. Afhankelijk van de verdere doelen van de sprint kunnen er ook nog andere deliverables zijn.

2.5 Testen

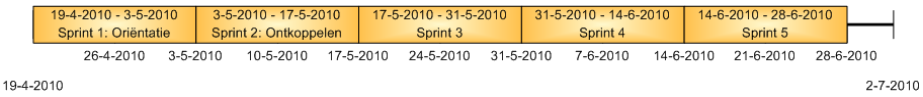
Een belangrijk onderdeel van het ontwikkelproces bestaat uit kwaliteit management. Bij E.Novation wordt de kwaliteit van ontwikkelde producten gecontroleerd door het Quality Assurance team. Daarnaast werken de ontwikkelaars volgens de Test Driven Development methodology.

Het ontwikkelteam schrijft, in TDD-stijl, van tevoren unit tests met JUnit en jMock. Voor integration testing wordt bijvoorbeeld de mailloop-check in de monitoring daemon gebruikt. Bij het ontwikkelen zullen ook wij gebruik gaan maken van JUnit voor Test-Driven Development.

Het QA-team doet voornamelijk acceptance testing van opgeleverde software: doet een systeem ook waar het voor ontworpen is? Gedurende ontwikkelsprints testen zij elke opgeleverde story. Bovendien wordt bij elke release gekeken of het geheel ook werkt.

Op basis van bijvoorbeeld use cases en specificatie documenten ontwikkelt het QA-team een test plan: wat moet er getest worden, en hoe? Vervolgens wordt een test beschrijving ontwikkeld. Daarin staan heel nauwkeurig de test procedures omschreven, in combinatie met de verwachte resultaten van die procedures. De resultaten van de test procedures worden opgenomen in het testrapport.

Afhankelijk van de voortgang van ons project en de definitieve wensen van E.Novation zullen we op enig moment QA laten kijken naar onze software. Na een demonstratie van een prototype kan QA wellicht acceptance tests ontwikkelen voor de door ons opgeleverde software.



Figuur 1: Tijdlijn

3 De doelen in de implementatiefase

In de tweede sprint zullen we ons richten op het neerzetten van een zelfstandige statistics en monitoring daemon. De functionaliteit van deze applicatie zal precies die functionaliteit omvatten, die de huidige in de EMS geïntegreerde daemons nu bieden.

We zullen in deze sprint in eerste instantie een complete ontwikkel- en testomgeving inrichten. Die hebben we nodig als sandbox om development, builds, packaging, deployment en testing van de nieuwe daemon te doen. Vervolgens gaan we de bestaande statistics collector en monitoring daemon onafhankelijk maken van de rest van de EMS. Als we dit gedaan hebben, zijn we waarschijnlijk al weer aan het eind van de sprint aangekomen (deze duurt in totaal slechts zeven werkdagen). Mochten we er tijd voor hebben, dan integreren we de monitoring en statistics functionaliteit ook meteen in één daemon.

In de drie sprints daarna zullen we de functionaliteit van deze nieuwe daemon gaan verbeteren en uitbreiden. Zo zullen we de mogelijkheid introduceren om checks te doen op verzamelde statistieken, en zullen we de nu actieve status checks van Nagios passief maken. De precieze indeling van de drie sprints weten we nu nog niet. Deze hangt af van de voortgang die we maken en de exacte prioriteiten van E.Novation, maar die worden pas aan het begin van iedere sprint vastgelegd in het Scrum model.