Eindverslag Bachelor Project Océ

Touch screen user interface design for Service

Vakcode
IN3405

TU Delft
Faculteit: Electrotechniek, Wiskunde en Informatica (EWI)

Gemaakt door:
Erwin Haasnoot 1372238
Jos Kuijpers 1311530

Commissie:
Begeleider Oce: Marvin Brunner

Begeleider TU Delft: Peter van Nieuwenhuizen

Coördinator BSc Project TU Delft: Peter van Nieuwenhuizen

Extra Commissielid TU Delft: Birna van Riemsdijk

Versie: 2.1

Voorwoord

Het laatste onderdeel van de Bachelor fase van de opleiding Technische Informatica aan de TU Delft wordt gevormd door het Bachelor (Eind)Project. Dit verslag is het eindrapport dat hoort bij dit project, dat in de vorm van een stage is gedaan bij Océ Technologies B.V. te Venlo. De opdrachtgever vanuit Océ is Marvin Brunner en onze begeleider c.q. coördinator van het Bachelor Project is Peter van Nieuwenhuizen. Wij willen Marvin Brunner en Peter van Nieuwenhuizen bedanken voor de begeleiding. Daarnaast willen wij ook al onze collega's bij Océ Technologies B.V. bedanken voor alle tijd die ze vrijgemaakt hebben om onze vragen te beantwoorden!

Samenvatting

Het onderhouden van printers is een complexe taak waar service monteurs speciaal voor opgeleid moeten worden. De technicus heeft een heel scala aan software tools tot zijn beschikking om deze taak beter uitvoerbaar te maken. Onder andere de Software Engineering 1 (SE1) afdeling binnen het Research & Development departement van Océ Technologies B.V. in Venlo werkt aan het verbeteren van deze software tools.

Voor het repareren en onderhouden van printers staan er drie grote software pakketten tot de monteur zijn beschikking, zogenaamde service software tools. Dit zijn het Service Diagnostic System (SDS), de Star tool suite en het Technical Service Manual (TSM). Alle drie deze software pakketten zijn aan vernieuwing toe en onder andere SE1 is druk bezig de nodige software aan te pakken.

In de huidige organisatie binnen Océ is het niet de taak van SE1 om bepaalde delen van de service software tools aan te pakken. Het doel van onze opdracht is daarom te kijken naar delen van de software pakketten waar bedrijfstakken binnen Océ goed samen kunnen werken (waar dat nu nog niet gebeurd) en te onderzoeken waar verbeteringen mogelijk zijn. Dit hebben we gedaan door ons eerst op de service tooling te oriënteren, waarna wij onszelf het beste zagen werken aan het TSM. Het TSM is een documentatie systeem waarin alle kennis die Océ heeft over zijn printers gepresenteerd wordt. Plannen voor ons nieuwe TSM systeem, genaamd DynamicTSM, zijn hierna uitgewerkt en deels geïmplementeerd.

Het uiteindelijke product is een prototype van het DynamicTSM systeem dat wij ontworpen hebben aan de hand van 5 verschillende ‘entry-points’. Dit zijn Diagnose door Error Codes, Diagnose door Symptomen, Preventive Maintenance, Modification en Installation. De 5 entry-points voor het DynamicTSM zijn ook de 5 redenen waarom een monteur bij een klant op bezoek gaat. Voor elk van de entry-points hebben wij ‘documentatie pagina’s’ ontworpen, waarvan alleen de Diagnose door error codes pagina’s geïmplementeerd zijn.

De informatie die nodig is om de pagina’s te vullen wordt uit de bron XML bestanden van het huidige TSM gehaald. Uiteindelijk is gebleken dat er geen 1 op 1 conversie mogelijk is van het normale TSM naar het DynamicTSM, omdat de bron XML niet goed gestructureerd is. Hierom hebben wij een aantal aanbevelingen gedaan die de structuur van de bron XML verbeteren.

Ten einde onze aanbevelingen te implementeren zullen er wijzigingen in de bron TSM aangebracht moete worden. Dit kan stapsgewijs gebeuren, bepaalde wijzigingen in de bron zullen bepaalde aanbevelingen mogelijk maken. Uiteindelijk zal er wel een redelijk deel van het TSM op de schop moeten om alle functionaliteit van ons DynamicTSM te kunnen benutten. Dit heeft als voordeel dat er, naast het implementeren van het DynamicTSM, nog veel meer gedaan kan worden met het TSM door de computer. Zeer waarschijnlijk ook functies waar wij niet eens aan gedacht hebben.

Definities en Afkortingen

TSM	Technical Service Manual
SDS	Service Diagnostic System
ITC	International Training Centrum
XML	eXtensible Mark-up Language
XSD	XML Schema Definition
DIP	Detailed Information Page
PP	Part Page
ToC	Table of Contents
UI	User Interface
SPM	Service Product Manager
HQ	Headquarters

Inhoudsopgave

Hoofdstuk 1 - Inleiding

Printers zijn complexe apparaten die vaak voor de core-business van bedrijven gebruikt worden. Het is daarom van belang dat printers goed functioneren. Elke minuut dat een printer het niet doet zorgt, vooral bij drukkerijen, voor verlies van tijd en, om het oude bekende gezegde maar te gebruiken, tijd is geld.

Voor het goed laten functioneren van printers is veel onderhoud nodig. Sommige onderdelen slijten na veelvuldig gebruik. Hier valt niks aan te doen, maar als deze onderdelen niet tijdig vervangen worden kunnen er veel ergere problemen optreden in een printer. Het is de taak van service monteurs om de printer goed te onderhouden en te repareren indien nodig.

Om de service monteur te helpen met zijn taak is er een pakket aan software tools, zogenaamde service software tools, ontwikkeld die het makkelijker moet maken om de printer werkend te houden. Een aantal jaar geleden is er onderzoek gedaan naar de mogelijkheden die voortkomen uit de integratie van deze tools. Hier is uit gebleken dat een technicus veel efficienter kan werken met deze tools naarmate ze beter geintegreerd zijn (Wang, 2007). Met dit onderzoek is uiteindelijk, naar ons weten, niets gedaan.

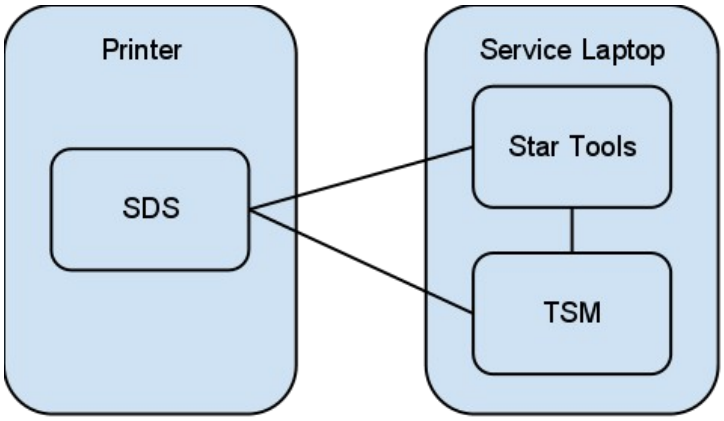
Wij hebben daarom van Océ de opdracht gekregen om naar dit service software tool pakket te kijken en een stukje eruit te pakken dat we kunnen verbeteren. Het is de bedoeling dat wij een ‘proof-of-concept’ maken, één die de samenwerking tussen het R&D departement en andere departementen binnen Océ naar een hoger niveau kan tillen. Dit zou de ontwikkeling van het softwarepakket, of in ieder geval het gedeelte waar wij aan werken weer in beweging kunnen brengen.

Uiteindelijk hebben wij gekozen om het Technical Service Manual aan te pakken. De probleemstelling die wij hanteren is: Het TSM helpt de monteur niet met het verkorten van service bezoeken. Punten waar wij ons op zullen richten, kunnen gevonden worden in appendix B.

Wij zullen in hoofdstuk 2 precies uitleggen wat het TSM is. Hoofdstuk 3 bevat het Programma van Eisen, welke requirements hebben we en aan welke hebben we wel en niet voldaan. In hoofdstuk 4 worden de tools aangeven die we gebruikt hebben om het TSM te verbeteren. In hoofdstuk 5 laten we zien hoe die eisen zijn verwerkt in het ontwerp.
Het proces dat we hebben doorlopen om ons doel te bereiken wordt beschreven in hoofdstuk 6. Als laatste geven wij in hoofdstuk 7 aan wat er structureel verkeerd is in het TSM om daarna in hoofdstuk 8 met aanbevelingen te komen om het TSM te verbeteren en hoe ons systeem zelf ook verbeterd kan worden.

Hoofdstuk 2 - Service Software Tools en het Technical Service Manual

Het deel van de service-software waar wij ons op moeten richten zit als volgt in elkaar; Elke monteur heeft een laptop bij zich, die op de printer aangesloten kan worden. Deze laptop bevat een software suite genaamd Star tools, alsmede een documentatie database genaamd het Technical Service Manual (TSM). Daarnaast wordt op de printer een web-applicatie genaamd het Service Diagnostic System (SDS) gehost, die op de Service Laptop gedraaid kan worden (in een browser). De browser is de client en de printer is de server. (Zie figuur 3.1)

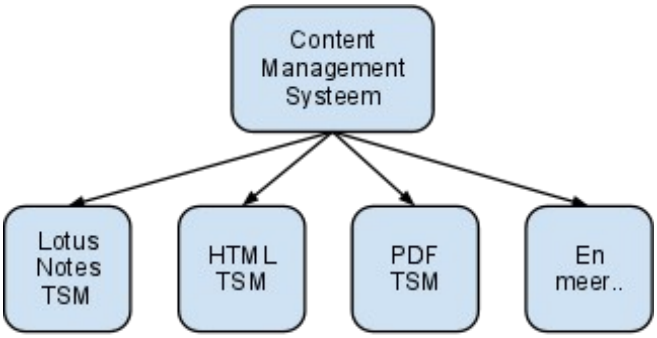


Figuur 3.1. Schematische weergave van Service Software Tools

Star tools is een software suite dat beheerd wordt door een software-groep binnen Océ HQ. Er zitten een aantal tools in deze software suite die regelmatig door service monteurs gebruikt worden om bijvoorbeeld bepaalde statistieken bij te houden over printers en op basis daarvan een werklijst aan te maken voor service monteurs. Tussen Star tools en het TSM en SDS is wel koppeling,maar hier is nog veel ruimte voor verbeteringen of zelfs integratie. Ontwikkeling van Star tools staat al een lange tijd op een laag pitje en qua organisatie is de Service bedrijfstak verantwoordelijk voor de ontwikkeling van dit software pakket. R & D kan er dus niet veel mee doen.
Uiteindelijk hebben wij besloten om het TSM aan te pakken. In dit hoofdstuk zullen we bespreken hoe deze in elkaar steekt, wat de achterliggende gedachte is en wat er naar ons idee mis is met dit TSM.

2.1 - Wat is Het Technical Service Manual?

Het Technical Service Manual bestaat uit twee delen, het ene deel is een database met alle onderhoud-kennis die Océ heeft over haar printers (de content). Het tweede deel is de presentatie van deze kennis. De content wordt opgeslagen in XML-bestanden en beheerd door een niet nader genoemd content management systeem (CMS). Dit CMS kan de content transformeren naar elke mogelijke presentatie vorm, denk hierbij aan HTML, PDF, en ook Lotus Notes.



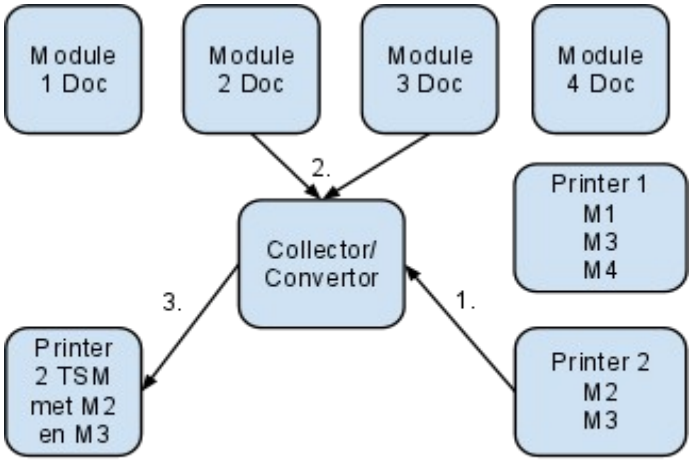
Figuur 3.2. schematische weergave van verbindingen tussen CMS en de verschillende TSM's

Op het moment wordt door monteurs van Océ gebruik gemaakt van de Lotus Notes TSM (LN-TSM) en monteurs van andere bedrijven gebruiken het HTML of PDF TSM. Deze TSM's zijn eigenlijk gelimiteerde vormen van het LN-TSM, en daarom zullen we ons hier richten op het LN-TSM.

2.2 - Modulaire Opbouw

Océ is een bedrijf dat veel verschillende soorten printers verkoopt en onderhoudt. Voor elke printer moet er een specifieke TSM zijn, die alle kennis bevat die Océ heeft over dat printertype. Tussen verschillende printertypes worden vaak modules hergebruikt. Twee zwart-wit printers die papier op verschillende manieren 'finishen' zullen dezelfde printer module gebruiken en verschillende 'finish'-modules. Deze printers zullen voor de printer module dezelfde installatie-procedures kennen en dezelfde problemen zullen voorkomen. Kortom, het is nodig dat er in de XML-TSM ook veel documentatie herbruikt kan worden. Het Internationaal Trainingscentrum (ITC), verantwoordelijk voor het up-to-date houden van het TSM, heeft dit als volgt geïmplementeerd: Voor elke printer wordt een printer-configuratie file bijgehouden. In deze file wordt opgeslagen welke modules een bepaalde printer heeft. In Figuur 2.2 is te zien dat er een printertype Printer 1 is met de modules 1, 3 en 4, daarnaast is
er een printertype Printer 2 met de modules M2 en M3.

Stel het is nodig dat het TSM van Printer 2 gegenereerd wordt in de vorm van een Lotus Notes database. Als eerste wordt de documentatie van verschillende modules verzameld. Aangezien wij hier geen officiële naam voor kennen noemen we dit proces "Collection"'; het uitvoerende programma noemen we de Collector. De Collector zal als eerste de printer-configuratie file uitlezen, hierin vindt hij dat Module 2 en 3 bij Printer 2 horen. Hij zal alle documentatie die bij deze 2 modules hoort bij elkaar zoeken en gebundeld presenteren aan de Converter. De Converter zal de XML bestanden uitlezen en in de vorm van een Lotus Notes database gieten.
De precieze details van hoe de Converter XML omzet in een Lotus Notes Database file is ons niet duidelijk geworden, en is ook niet relevant voor ons project.



Figuur 3.3. Schematische weergave van hoe Printer specifieke TSM 's gemaakt worden.

2.3 - Distributie

Distributie van het TSM gebeurt, in het geval van Lotus Notes TSM, door middel van nachtelijke 'Pushes'. Service Laptops met het TSM worden elke avond aan een poort gehangen waarna de nieuwste versie van het TSM automatisch geüpload wordt naar de laptops. Dit duurt ongeveer een uur, aangezien van elk type printer het TSM overgezet moet worden, wat in totaal om een paar gigabyte aan data gaat. Andere TSM's (o.a. PDF) hebben geen reguliere updates en worden op verzoek van klanten verspreidt.

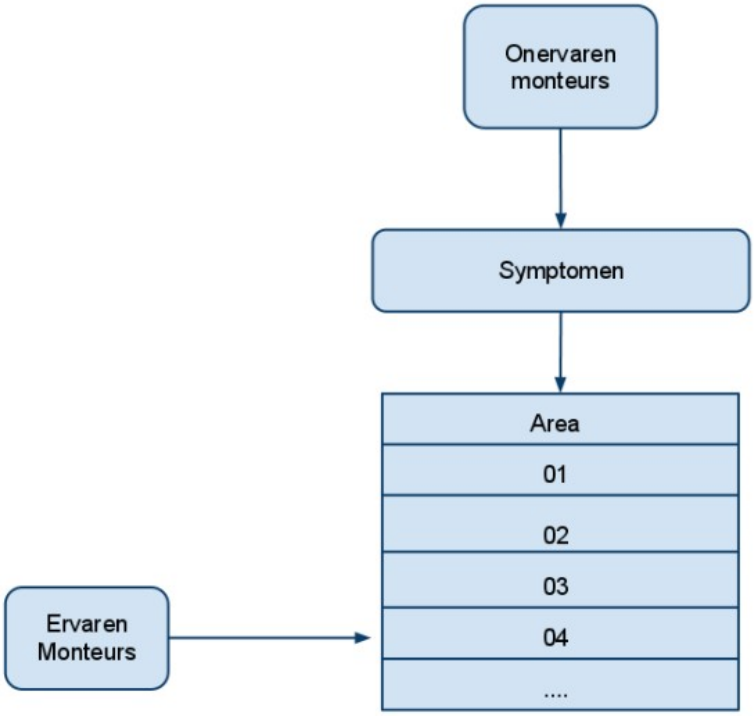
2.4 - Annotaties en Aanpassingen aan het TSM

Een van de belangrijkste features van Lotus Notes is het gebruik van annotaties. Annotaties zijn aanpassingen die de gebruiker van het TSM aan documenten kan maken om een fout te corrigeren of

zelf een aantekening te maken. De gebruiker kan ervoor kiezen om zijn annotatie door te spelen aan zijn lokale Service Product Manager (SPM), indien hij de annotatie belangrijk genoeg acht. De SPM zal vervolgens reviewen of deze annotatie belangrijk genoeg is om naar andere technici binnen zijn Operating Company (OpCo) door te spelen, of zelfs gelijk door te sturen naar HQ (indien erg belangrijk). Een SPM is over het algemeen iemand die heel veel kennis heeft van printers van Océ en als zodanig in een positie is om deze annotaties op waarde te beoordelen. Het ITC zal, in opdracht van HQ, deze annotaties verwerken in het systeem en deze aangepaste documenten aanmerken als 'Review'. Hierna zal een SPM controleren of de informatie in het document juist is. Als dit het geval is, dan zal de SPM het document aanmerken als 'Ready to Release', waarop het document in de volgende push meegenomen zal worden. Documenten die gepusht zijn worden aangemerkt als 'Released'.

2.5 - Lotus Notes TSM Structuur

Het ITC houdt zich bezig met het bruikbaar maken en houden van het TSM, voor zowel ervaren als onervaren monteurs. Ze doen dit door 2 entree-punten in de documentatie te bouwen, zoals te zien is in figuur 3.4 voor diagnose van printers. Dit systeem wordt binnen het hele TSM toegepast. De "vertical entry" wordt gebruikt door onervaren monteurs. Zij komen na een gesprek met de klant achter een aantal symptomen van het probleem. Door middel van de Symptom classification in het TSM zoeken zij stapsgewijs naar een symptoombeschrijving die past bij de symptomen zoals de klant ze aan de monteur heeft verteld. Uiteindelijk zullen ze, na het volgen van de pagina's in het TSM, uitkomen bij het juiste onderdeel (Area) van de printer, bijvoorbeeld area 04.
Een ervaren monteur heeft een gesprek met de klant, en kan op basis van zijn ervaring direct de juiste diagnose stellen. In dit geval zal hij direct door hebben dat het probleem wel in area 04 moet zitten en zal via zogenaamde "horizontal entry" in 1 stap naar de juiste pagina gaan, waarna de informatie die daar staat hem helpt met het repareren / lokaliseren van het onderdeel. Hierbij wordt wel opgemerkt dat zeker de meer ervaren monteurs minder gebruik maken van de TSM en vanuit hun ervaring direct (kunnen) gaan sleutelen om het probleem op te lossen.



Figuur 3.4. Schematische weergave van hoe ervaren monteurs TSM gebruiken tegenover onervaren

2.6 - Document Templates

Alle pagina's binnen de TSM krijgen vanuit de XML-database een vaste structuur (template), zodat ze qua structuur consistent zijn. Er zijn verschillende soorten pagina's en elk type pagina heeft een eigen template, bijvoorbeeld een template voor installatie-pagina's en diagnose pagina's. Deze templates zijn gevormd uit een aantal velden waarin bepaalde blokken informatie geplaatst kunnen worden. Voor deze templates wordt veelvuldig gebruik gemaakt van het hergebruiken van blokken tekst, een gevolg van de modulaire opbouw.
De volgorde van de velden ligt vast en kan niet gewijzigd worden. Er kan wel voor gekozen worden om bepaalde blokken weg te laten als die voor een specifieke pagina niet nodig zijn, maar elk type blok kan

maar op exact 1 plaats neergezet worden binnen de pagina. Bijvoorbeeld het description block, bij een diagnose pagina, zal altijd bovenaan staan en dus zal de description zelf altijd bovenaan staan.

2.7 - Verbeteringen aan het TSM

Om het TSM te kunnen verbeteren moeten we eerst vast stellen wat de verbeterpunten zijn. Na gesprekken met de makers (ITC & service afdeling), de gebruikers (Service Technici) en buitenstaanders (o.a. de Research & Development afdeling) hebben wij hier een beeld van kunnen maken. Om het behapbaar te houden zullen we het hier vooral hebben over de presentatie van het TSM, dus wat er verbeterd kan worden aan het TSM zoals die nu in Lotus Notes is. De voorgestelde verbeteringen zijn in het vervolg van deze paragraaf telkens cursief weergegeven.

Aan het gebruik van Lotus Notes kleven een aantal nadelen, zoals bevestigd door Océ (Tunnissen, 2011) Lotus Notes is niet portable naar andere systemen (smartphones, tablets) en heeft hoge licentiekosten. Deze hoge kosten zorgen er voor dat andere bedrijven die service willen leveren aan Océ-printers, maar geen Lotus Notes licentie hebben, het moeten doen met PDF of HTML TSM's. Deze TSM's missen een groot gedeelte van de functionaliteit die het Lotus Notes TSM wel heeft.

Maak het nieuwe systeem in een vrije omgeving (HTML, Java) zodat er geen licentiekosten meer zijn en het systeem een stuk beter portable wordt.

Het eerste wat opvalt bij het gebruik van het TSM in Lotus Notes is de Table of Contents (ToC). De structuur van de ToC bestaat, zoals beschreven in paragraaf 2.5, uit area's en sub-area's. Een gevolg hiervan is dat pagina's een aantal lagen diep zitten, en er redelijk wat klikwerk nodig is (ook voor ervaren monteurs) om via de 'verticale' entry een pagina te bereiken. Het gebruiken van de zoekfunctie werkt ook niet goed, vooral omdat er niet goed met typefouten omgegaan wordt.

Directe links vanuit bijvoorbeeld het SDS naar het TSM zouden dit probleem gedeeltelijk kunnen oplossen, in ieder geval voor het oplossen via Error Codes.

Als we dan een aantal pagina's openen valt ons gelijk de tab-structuur op van Lotus Notes. Elke aparte link opent namelijk een nieuwe tab. Vooral als je een instructie volgt (bijvoorbeeld het installeren van een printer) raak je snel de weg kwijt zodra je begint te klikken op links binnen links binnen links. Naast het minimaliseren van het aantal links in een pagina merkten wij dat er ook veel 'pre-requisites'[1] in de instructies staan, met links naar de pagina's die uitleg geven over hoe hieraan te voldoen. Bijvoorbeeld, installeer onderdeel 12 voordat je onderdeel 13 installeert, met een link naar hoe je onderdeel 12 installeert.

De TSM zou alle 'pre-requisites' automatisch kunnen checken en afvinken als ze al voldaan zijn, of in dit geval verbergen voor de gebruiker.

De TSM zou ook bij moeten kunnen houden welke stappen al uitgevoerd zijn. Knipperende links (functionaliteit van Lotus Notes) voldoen hier niet voor

Als we dan kijken naar diagnose pagina's zien we meerdere punten van verbetering verschijnen. Elke diagnose pagina is opgebouwd uit de volgende blokken: Description, Tests & Checks en Suspected Parts & Settings.
Het Suspected Parts & Settings block is een tabel met in elke rij een 'suspect'. Dit zijn onderdelen of settings die mogelijk de oorzaak van het probleem, die de diagnose pagina beschrijft, kunnen zijn. Tests & Checks bevat een lijstje met testen die de monteur uit kan voeren om duidelijker te maken welke suspects waarschijnlijk de oorzaak zijn van het probleem.

[1] Een pre-requisite is een stuk hardware of software dat voor een bepaalde stap al geïnstalleerd moet zijn.

De volgorde van de suspects tabel wordt momenteel gebaseerd op de tijd die het kost om de bijbehorende checks uit te voeren. Hierdoor kan het voorkomen dat suspects die het meest voor een bepaalde error zorgen helemaal onderaan komen te staan in de lijst.

De volgorde zou dynamisch aanpasbaar moeten zijn, zodat je de positie van een suspect in de lijst kan berekenen. Deze berekening kan bijvoorbeeld gebeuren aan de hand van al aanwezige statistiek over hoe vaak een suspect de oorzaak is van een probleem.

Het Test & Checks block is daarnaast aan een verbetering toe wat betreft de computer-uitleesbaarheid. Voor ons (als mens) is het al lastig om te achterhalen welke suspects je checkt bij een bepaalde test, voor een computer zonder natural language parser is het totaal onmogelijk.

Om het voor een monteur, en voor de computer, duidelijker te maken zouden de parts expliciet genoemd kunnen worden bij elke Test & Check. Als er een Test is zonder bijbehorende suspects, waarom staat die check dan daar? Als diezelfde test dan verwijst naar een suspect die niet in de lijst staat, waarom staat die suspect niet in de lijst?

Daarnaast zou de monteur kunnen aangeven of een test gelukt is of niet, waarna de computer de volgorde in het Suspected Parts & Settings block zou kunnen aanpassen.

Als je verder kijkt in het Suspected Parts & Settings block zie je een subtiel verschil tussen suspects. Sommige suspects hebben een link naar een zogenaamde 'partslist' en sommige niet. Dit leidt ons er toe te vermoeden dat het verschil tussen Parts & Settings een link naar een partslist is.

Maak expliciet welke entries parts zijn en welke entries settings zijn, zodat een systeem hier op andere manieren mee om kan gaan.

Waarom suspects limiteren tot parts en settings? Een suspect kan ook zijn 'papier zit niet goed in de lade'. In theorie een suspect, maar heeft geen plek in deze tabel. Hier zou ook statistiek over verzameld kunnen worden.

Als je dan op de partslist komt, zie je bovenin een plaatje staan met veel nummertjes. Deze blijken overeen te komen met de parts in de partslist, de index nummers van parts is terug te vinden in het plaatje, zodat je weet waar de parts zich bevinden in de printer. Geen mogelijkheid tot inzoomen en veel nummers die dicht bij elkaar staan zorgen ervoor dat het lastig is om een part precies te localiseren op zo'n plaatje.

Zorg voor een inzoom functie.

De verbeterpunten die hier besproken zijn, zijn de punten die wij onder andere aan willen pakken met ons systeem. Een korte samenvatting:

<u>TSM:</u>

- Licentie Lotus Notes is duur.
- Geen portabiliteit tussen hardware-platformen naast PC's.
- Knipperende links zijn enorm irritant en leiden tot vroegtijdig klikken
- Tab-structuur niet altijd wenselijk, elke aparte link opent een nieuwe tab. Als er vanaf 1 bepaalde pagina een diepte[2] van 5 pagina's is, raak je heel snel de weg kwijt. Mede mogelijk gemaakt door de absentie van een manier om de gevolgde stappen bij te houden.
- Geen manier om bij te houden op welke stap je precies zit, als je een stappenplan aan het volgen bent.

[2] Diepte in deze context - Het aantal clicks dat nodig is om vanuit een pagina naar een andere pagina te komen. Ervan uitgaande dat er geen cirkels zitten in de linking. Dat het niet mogelijk is om vanuit een pagina via andere pagina's naar die ene pagina terug te komen.

- Zoekopdracht werkt niet goed. De zoekopdracht moet exact kloppen, als er aan het begin en/of eind karakters / cijfers missen, wordt er niets gevonden.
- Weergave van volgordes, bijvoorbeeld bij de lijst met onderdelen die te maken kunnen hebben met een bepaald probleem of een lijst die je krijgt als je zoekt, is gedeeltelijk Random (zit geen duidelijke logica achter).
- Tests & Checks geven niet duidelijk aan op welke suspects zij van toepassing zijn.
- Er is geen link tussen het Test & Checks block en het Suspected Parts & Settings block
- Er is geen mogelijkheid tot inzoomen op onderdeel-plaatjes. Lastig om onderdelen exact te lokaliseren als er heel veel onderdelen in een klein gebied zitten.

Hoofdstuk 3 - Programma van eisen

3.1 Huidige Situatie

Voor het servicen van printers gebruiken service monteurs een drietal tools om een probleem te kunnen identificeren:
1) SDS-tests
2) STAR-Tools
3) TSM

In de huidige situatie bevinden deze tools zich op een laptop die de service monteur bij zich heeft. Echter, het kost tijd om zo'n laptop aan te sluiten, op te starten en de software te starten die nodig is. Daarnaast komt het voor dat de laptop (of de software) crasht en dan kost een reboot nog meer tijd. Dit kan er toe leiden dat een service monteur direct gaat sleutelen, terwijl het wenselijker is om eerst de data te bekijken.

Daarnaast is er maar beperkte integratie tussen de drie tools.

3.2 Gewenste Situatie

Om de service te kunnen verbeteren zijn er een aantal dingen die er voor kunnen zorgen dat het proces soepeler verloopt en er minder tijd verloren gaat:

1) Integratie SDS met TSM en STAR
In de huidige situatie is er slechts beperkte integratie tussen de verschillende tools. Zo wordt de feedback van een SDS-test niet doorgegeven aan bijvoorbeeld het TSM om een bepaalde pagina of sectie (Suspected Parts & Settings) weer te geven. Het is wenselijk om de integratie van de tools te bevorderen, zodat een Service Technicus zich meer kan focussen op het oplossen van problemen en niet om moet kunnen gaan met allerlei verschillende tools.

2) Het afstappen van Lotus Notes en het TSM te draaien op de printer zelf.
Door SDS te integreren met TSM wordt het mogelijk om beide op de printer zelf te draaien, waardoor de service laptop minder noodzakelijk is en er tijdwinst geboekt kan worden. Voor de STAR-Tools moet nog een alternatieve oplossing bedacht worden. De focus van ons project ligt op integratie van de TSM en de printer (en daarmee ook met de SDS).
Daarnaast kan de licentie van Lotus Notes afgebouwd worden en hoeft er minder rekening gehouden te worden met bedrijven die deze licentie niet hebben (dan hoeft er geen aparte TSM gemaakt te worden, in bijvoorbeeld HTML).

3) Het TSM zelf
Momenteel wordt er absoluut geen gebruik gemaakt van beschikbare statistieken rond onderdelen die meestal verantwoordelijk zijn voor het veroorzaken van bepaalde errors. De volgorde van de Suspected Parts & Settings is bijvoorbeeld gebaseerd op de tijd die het kost om een bepaalde test te runnen. Als er gebruik gemaakt wordt van statistiek kan de monteur direct de "waarschijnlijke veroorzaker" testen en zo tijd besparen. De huidige situatie geeft een TSM waarin alle pagina's statisch aangemaakt zijn en weergegeven worden. In de gewenste situatie wil je een pagina pas opbouwen wanneer deze opgevraagd wordt (ruimte besparing) en dat deze dynamisch aanpasbaar is om feedback te kunnen verwerken vanuit een SDS-Test of een Test & Check block.

3.3 Requirements

Alle requirements zoals wij die hebben opgemerkt / hebben gekregen / bedacht hebben, staan in het Requirements Document in appendix C. Er moet wel rekening gehouden worden met het feit dat, om dit project volledig uit te voeren, een periode van 2 maanden te kort is. Er zijn dus veel requirements opgesteld die niet aan bod komen binnen ons project, puur omdat er te weinig tijd voor was. Bij het

opzetten van de requirements is geen onderscheid gemaakt tussen de verschillende prioriteiten ze horen te hebben. Vanwege de omvang van het systeem is het lastig om een goed overzicht te maken van de prioriteiten. Nadat de keuze gemaakt was om ons te richten op “Diagnosis through Errors” lag de focus voornamelijk op Requirements die hier mee te maken hadden.

Om een overzicht te geven staan hieronder een tweetal (voor ons) belangrijke requirements die gehaald zijn en een tweetal even belangrijke requirements die niet gehaald zijn. Bij de genoemde requirements volgt een uitleg (cursief) waarom de requirement belangrijk is. Tussen haakjes staat achter iedere requirement uit welke categorie van het Requirements Document deze afkomstig is.

3.3.1 Wel gehaalde requirements

- Vanaf een errorcode direct naar de juiste error pagina doorverwezen worden. (Algemeen)
  In de huidige TSM structuur moet een Service Monteur zelf de pagina opzoeken die hoort bij een bepaalde error code. Een probleem dat hierbij op de loer ligt is bijvoorbeeld typefouten. Daarnaast kost het tijd om iets zelf op te zoeken. Praktisch de meest belangrijkste requirement was de directe link naar een Error Pagina. Dit scheelt tijd en voorkomt typefouten.

- Lijst met Suspected Parts & Settings geordend weergeven, gebaseerd op statistieken van de parts en mogelijk uitkomsten van test/checks. (Detailed Info Page)
  In de huidige TSM wordt de volgorde van Suspected Parts & Settings bepaald aan de hand van de tijd die het kost om een bepaalde test uit te voeren. Om er voor te kunnen zorgen dat een Service Visit zo min mogelijk tijd kost, is het belangrijk dat de volgorde gebaseerd is op de kans dat een bepaald onderdeel de “boosdoener” is. (Note: de benodigde statistieken zijn beschikbaar).

3.3.2 Niet gehaalde requirements

- Test/checks moeten weergegeven worden en de uitkomst van zo’n Test moet verwerkt worden in de volgorde van Suspected Parts & Settings. (Detailed Info Page)
  Deze requirement is deels wel en deels niet gerealizeerd. Het was lastig om de benodigde informatie te verkrijgen uit de XML met als gevolg dat er slechts bij 1 pagina een Test & Check block gedefinieerd is (door de informatie handmatig in een XML format te zetten). Echter wordt de feedback wel verwerkt in de volgorde van Suspected Parts & Settings.

- Statistiek (trend-analyse) over errors kunnen zien. Hoe vaak is een bepaalde error opgetreden en op welke tijdstippen. (Detailed Info Page)
  Dit is een functionaliteit die momenteel aanwezig is in de STAR-Tools, maar deze informatie kan prima in een Dynamisch TSM systeem verwerkt worden. Ons systeem bevat een simpele mechaniek voor statistiek en deze kan met een kleine aanpassing voor trend-analyse gebruikt worden.

- Annotaties aanbrengen op een pagina, of aangeven waar een fout staat (Algemeen)
  Om fouten in een TSM door te kunnen geven wordt in het huidige Lotus Notes gebruik gemaakt van annotaties. Dit is een prima vorm om feedback te geven en praktisch om te behouden.

Hoofdstuk 4 - Tools

Java

De taal die gebruikt is voor het project is Java. Hier is voor gekozen aangezien het SDS systeem waar we ons systeem op willen aansluiten ook gebouwd is in Java. Dit zou het makkelijker moeten maken om de integratie tussen de twee onderdelen (SDS en TSM) te kunnen maken. Voor de programmeeromgeving is gekozen voor Eclipse aangezien we deze IDE allebei gewend zijn.

WindowBuilder (Pro)

Om snel GUI's te kunnen prototypen is een (swing) GUI designer erg belangrijk. Het laat je focussen op de look-and-feel van de GUI, in plaats van de code die de GUI beschrijft. GUI designers die integreren met Eclipse (plug-ins) hadden onze voorkeur, zodat een project niet vanuit meerdere plekken beheerd hoeft te worden. WindowBuilder Pro bleek een Eclipse plug-in te zijn die beheerd wordt door Google en aangezien wij goede ervaringen hebben met Google-producten besloten we WindowBuilder ook te gebruiken.
Uiteindelijk bleek dat het, naast het opzetten van een basic GUI, voor ons praktischer was om de code uit te schrijven dan een GUI designer te gebruiken, maar al doende leert men.

XML
Om ons project een zekere mate van validiteit te geven, is het nodig dat wij een huidige vorm van het TSM kunnen parsen om op basis daarvan onze pagina's aan te maken. Hiervoor hebben wij de bron-bestanden van het HTML-TSM (HTSM) gebruikt, wat een 1-op-1 conversie is van het TSM dat in Lotus Notes draait). Deze bron-bestanden zijn XML bestanden, waar nogal een aantal regels overtreden worden van de XML-recommendation zoals opgesteld door w3c. Er wordt bijvoorbeeld veel layout gebruikt in deze 'XML'-bestanden. Bijvoorbeeld: het precies, tot aan de pixel, definiëren van een tabel in de error message pages.
Omdat deze XML bestanden nodig aan verbetering toe waren, besloten wij om XML te gebruiken voor een soort tussenvorm-XML bestanden. (Die wel aan de w3c recommendations voldoet). Deze bestanden kunnen wij dan gebruiken om een aantal aanbevelingen meer kracht bij te zetten.

XSLT en XPATH

De transformatie besproken bij het XML kopje werd eerst gedaan in Java code met behulp van DOM-trees. We wisten al hoe we deze DOM-tree moesten gebruiken, omdat we al de conversie van dummy pagina's naar GUI elementen hadden gemaakt ermee. (Dummy pagina's geparsed, JSwing components aangemaakt).
De Java code die hieruit rolde was, naarmate er meer 'speciale' cases afgevangen moesten worden, steeds minder makkelijk om te lezen. Na een tijdje gezocht te hebben naar een mogelijke vervanger vonden wij XSLT (eXtensible Stylesheet Language Transformations) (en XPATH), nog een w3c recommendation. Het grote voordeel van XSLT is dat het werkt op basis van pattern matching (Tidwell, 2008). Wat in Java geschreven werd met vele nested if-statements en for-loops kon in XSLT d.m.v. XPATH makkelijk en duidelijk in 1 regel beschreven worden. Hierdoor werd het afvangen van speciale cases, waar er veel van zijn, veel makkelijker.
Daarnaast is er in de vorm van Saxon in Java een goede implementatie van een XSL transformer beschikbaar.

Al deze voordelen hebben er ons toe gezet om van de transformatie in Java code af te stappen en XSLT te gebruiken.

XML Schema (XSD)

Wij hadden een manier nodig om de structuur van de tussenvorm-XML bestanden te beschrijven. Hiervoor vonden wij 2 opties, Document Type Definition (DTD) en XML Schema Definition (XSD). Dit zijn allebei talen die de structuur en content van XML bestanden kunnen definiëren en handhaven. Het grote verschil tussen DTD en XSD is dat XSD een vorm van XML is, wat alle voordelen met zich meebrengt die normale XML-bestanden ook hebben. (XSL transformaties, (recursieve) Schema definities). DTD voldoet niet aan de eisen van een XML-file, waardoor deze door velen (van der Vlist, 2002) als minder krachtig beschouwd wordt. Wij hebben daarom gekozen om XSD te gebruiken voor al ons definitie-werk.

Hoofdstuk 5 - Ontwerp

Het ontwerp van het DynamicTSM systeem is te vinden in appendix D. In dit hoofdstuk worden de onderdelen uitgezet die niet geïmplementeerd zijn, maar die wel belangrijk zijn voor het systeem en voor de Service Technicus.

Belangrijk om te weten is dat elke printer, waar we DynamicTSM voor ontworpen hebben, een eigen computer heeft. Dit, inclusief een beeldscherm van 1024 bij 768 pixels.

De entry-point “Diagnosis through errors” is tijdens de implementatiefase geïmplementeerd. Voor de andere 4 entry-points is er een ruw design gemaakt, de een wat meer uitgebreid dan de andere, maar het valt daadwerkelijk uit te werken en toe te voegen tot het DynamicTSM. Of beter gezegd, het DynamicTSM is pas af als deze 4 entry-points zijn toegevoegd.

De volgende paragrafen beschrijven de andere 4 entry-points.

5.1 Symptom Classification

Problemen bij de printer komen niet alleen voort uit errors die optreden, waarbij een Error Pagina de uitkomst moet bieden. Andere problemen, zoals “Image Quality”, zijn veel lastiger om een specifieke pagina voor te maken. Dit komt omdat er veel symptomen zijn die allemaal een andere oorzaak kunnen hebben. We hebben de keus gemaakt uit praktisch oogpunt om deze “ingang” niet mee te nemen tijdens de implementatie. Wel is er een algemeen ontwerp gemaakt over hoe dit punt in onze visie gerealiseerd zou moeten worden. Hiervoor wordt de huidige methode deels hergebruikt en deels anders opgezet.

In onze opzet wordt de boom minder breed, maar wat dieper. In het huidige systeem werkt Symptom Classification voor bijvoorbeeld “Image Quality” als volgt: Er zijn een aantal categorieën gedefinieerd en daarbinnen worden alle mogelijke varianten weergegeven. Dit geeft een lange lijst met voorbeelden waar de Service Technicus langs moet gaan.

In een alternatieve opzet worden dezelfde categorieën gebruikt, maar wordt wat daar onder ligt verder uitgesplitst. Dit zou als gevolg moeten hebben dat de Service Technicus niet meer op een pagina met meer dan tien voorbeelden terecht komt, maar via navigatie de selectie steeds kleiner kan maken. Echter moet hier wel bij gezegd worden dat Symptom Classification het meest complexe probleem is om op te lossen voor de opbouw van een TSM. Error Classification is heel recht-toe-recht-aan, maar Symptom Classification is veel vager. Daardoor is het moeilijk om een systeem te bedenken dat het altijd en efficiënt op zal lossen. Hierdoor is er ook besloten om dit onderdeel niet verder uit te werken op dit moment.

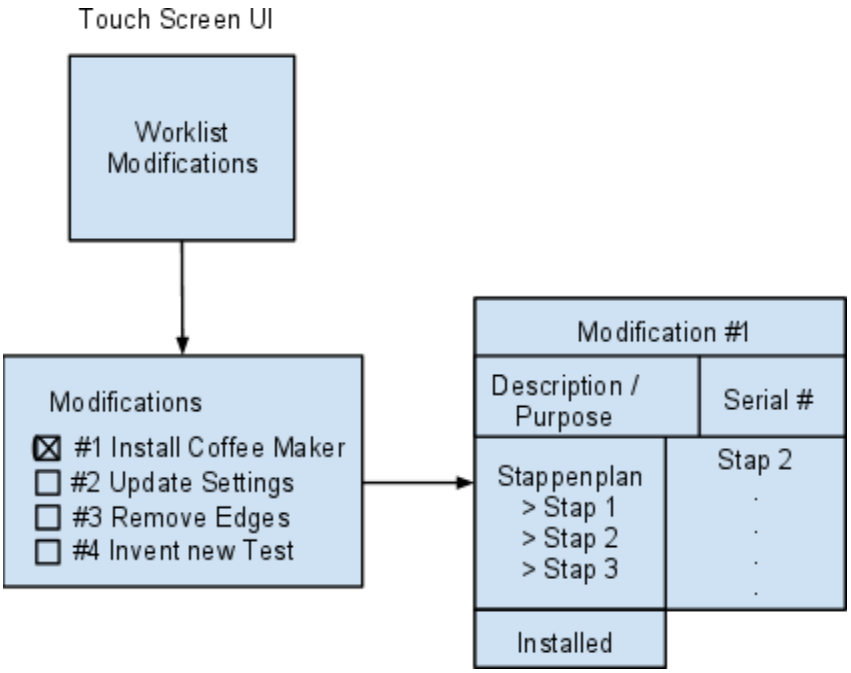
5.2 Installation Pages

Voor de installatie pagina(‘s) is er gekeken naar hoe deze in elkaar zit. Een installatiepagina bestaat uit een serie stappen die doorlopen moet worden, welke onderverdeeld zijn in meerdere substappen die elk vaak een eigen pagina hebben (installatie pagina voor een deel van de printer). In de huidige TSM-vorm in Lotus Notes heeft dit tot gevolg dat er een hele serie tabs opengezet wordt en dan kan het overzicht verdwijnen. Er is geen manier om bij te houden waar je bent gebleven en dat is dan ook het enige punt waar wat op te verbeteren valt. Mede door de beperkte tijd en aangezien het slechts een klein onderdeel vormt van de gehele TSM (er zijn veel meer pagina’s gelinkt aan errors, symptomen, enz) is er voor dit onderdeel een simpele uitbreiding opgezet:

Het installatieschema (dat nu in tabelvorm staat) overzetten naar een structuur waarbij een Table of Contents is toegevoegd aan de zijkant (soortgelijk met de ToC van de Modifications (Zie Figuur 6) en die van de Error Pages, die beide op dezelfde manier opgezet zijn. De Table of Contents die toegevoegd wordt, blijft te alle tijden zichtbaar tijdens het proces, zodat de Service Monteur kan springen naar het gewenste punt, ook als hij op een later tijdstip verder gaat. De installatie blijft lineair, maar de Service Monteur kan kiezen welke pagina’s hij wel en niet nodig heeft.

5.3 Modifications

Naast het afhandelen van problemen zijn er ook nog de Modifications die afgehandeld moeten worden. Deze worden normaliter weergegeven in de vorm van een lijst met modifications die uitgevoerd kunnen worden op de printer. Hierbij wordt echter geen rekening gehouden met het feit dat bepaalde modifications niet van toepassing zijn op een bepaalde range van serienummers van een type printer. Het is wenselijk dat voor een bepaalde printer alleen dié modifications getoond worden die ook daadwerkelijk van toepassing zijn op de printer, afhankelijk van het serienummer. Bij elke modification is een range van serienummers meegegeven waarop deze modification van toepassing is. Voor ons design wordt daar rekening mee gehouden en worden alleen de modifications weergegeven die nuttig zijn (die dus binnen de range vallen)..
Zoals in onderstaande figuur (Figuur 8.5) staat weergegeven kan er vanuit de worklist doorgelinkt worden naar de Modifications Page die dit overzicht heeft. Hier kan dan een bepaalde Modification geselecteerd worden en wordt de bijbehorende pagina geopend. Deze bevat een algemeen descriptionblock net zoals de Error Pages en heeft een stappenlijst waarin stapsgewijs besproken wordt wat er gedaan moet worden. Zodra alle stappen zijn uitgevoerd kan er via de “Installed” button aangegeven worden dat de Modification is uitgevoerd.



Figuur 8.5 - Schematische weergave van Modification GUI

5.4 Preventive Maintenance

Elke part valt onder één van 3 categorieën: Dit zijn parts die in principe niet kapot gaan, parts die soms kapot gaan en parts die een beperkte levensduur hebben en dus regelmatig vervangen moeten worden. Vooral op deze laatste 2 categorieën is Preventive Maintenance gericht: het tijdig vervangen van onderdelen om errors te voorkomen. Voor dit soort parts worden een tweetal waarden bijgehouden in de printer: Het aantal prints dat gemaakt is en het aantal prints dat de part zonder problemen zou moeten kunnen maken. Daarnaast wordt er bijgehouden wanneer een bepaalde part vervangen is (op welk percentage van de levensduur). Vanuit de datalogs kunnen dit soort gegevens opgehaald en bepaald worden. In Figuur 8.6 staat een voorbeeld van zo’n tabel zoals deze opgebouwd zou kunnen worden.

partname	lifecounter	current counter	%	avg %
a	500	125	25	80
b	500	110	22	75
c	500	250	50	77
d	500	300	60	91
e	500	475	95	94

Figuur 8.6 - Schematische weergave van Preventive Maintenance GUI lijst

Het is praktisch om vanuit het SDS (de gehele tabel) of de TSM (waarden voor bepaalde parts) deze informatie op te kunnen vragen, aangezien het aantal parts met een lifecounter beperkt is, zal de tabel een beperkte grootte hebben. Hiermee krijgt de Service Technicus een goed overzicht van de parts die binnenkort vervangen moeten worden en dan kan hij beslissen om zo'n part direct te vervangen, zodat dit een service visit scheelt. Verder bevat het een link naar de bijbehorende Part Page (vanuit de Part Name), zodat de Service Technicus snel bijbehorende informatie kan opvragen over bijvoorbeeld Disassembly instructies, Diagrammen, enz. In de huidige TSM is dit niet mogelijk vanwege het ontbreken van Part Pagina's en moet de Service Technicus handmatig deze informatie opzoeken en dat kost meer tijd. Door informatie uit verschillende bronnen, zoals werklijsten en datalogs, te bundelen kunnen dit soort overzichten gemaakt worden.

Hoofdstuk 6 - Proces

In appendix B, hoofdstuk 5 is de algemene planning opgenomen die gebruikt is tijdens het project. In dit hoofdstuk wordt er wat meer verteld over wat er per week gebeurd is. Over wat voor problemen we tegen zijn gekomen en hoe daar mee is omgegaan.

Week 1
De eerste week bij Océ was voornamelijk een week waarin er veel tijd doorgebracht is met rondvragen bij de verschillende mensen binnen de projectgroep waar we bij aangesloten zijn. De licentie die nodig was om de TSM te kunnen bekijken kwam pas in de 2e helft van de week en vanaf donderdag konden we daadwerkelijk gaan beginnen met het onderzoek naar de TSM.

Week 2
De tweede week begon met een afspraak met de begeleider van de TU Delft (Peter van Nieuwenhuizen). Daar zijn een aantal afspraken gemaakt rond het inleveren van documentatie e.d. In deze week is ook de opzet voor Plan van Aanpak en Oriëntatieverslag gemaakt en deze zijn ingeleverd. Daarnaast is er een meeloopdag geweest met een Service Technicus waarbij ook extra inzichten verkregen zijn rond ervaringen van de Technici met de TSM.

Week 3-4
De weken 3 en 4 zijn gebruikt om het design grotendeels af te maken. Daarvoor zijn de Requirement Doc, de verschillende Sequence Diagrammen gemaakt en de eerdere docs (Plan van Aanpak en Orientatieverslag) aangepast aan de hand van de feedback. In deze periode zijn we ook op het ITC langs geweest voor een tweede afspraak en daarmee is meer inzicht verkregen rond de XML opzet van de TSM en wat er mee mogelijk is.

Week 5-6
In week 5 is de implementatiefase begonnen met de eerste opzet van de GUI. In eerste instantie is er gebruik gemaakt van Dummy bestanden om dingen te visualiseren, later in het project zal dit omgevormd worden naar daadwerkelijke TSM files. Het merendeel van de GUI is in deze 2 weken geïmplementeerd. Verder zijn zaken als een Table of Contents en de algemene pagina's (Detailed Info Page, Part Page, e.d.) aangemaakt. In week 6 is er overgegaan van Dummy pagina's naar het inlezen van de XML files van de HTML-TSM (HTSM). Verder zijn de Design Doc en Requirements Doc gestuurd naar de begeleider voor feedback.

Week 7-8
Aangezien het inlezen van de Error Pages grotendeels werkt (XML geeft nog steeds problemen) is er gewerkt aan de nieuwe functionaliteit: Statistiek. Aan de hand van het aantal keer dat een Part in verband is gebracht met een Error, wordt de volgorde van de Suspected Parts & Settings (die nu "hard-coded" is gedefinieerd in de XML) aangepast om de statistiek te verwerken. Verder is er een eerste implementatie voor het laatste block van de Error Page toegevoegd: het Test & Check Block. In week 8 is het Test & Check Block uitgebreid om feedback te sturen naar het Suspected Parts & Settings Block. De volgorde van de suspects wordt nu ook aangepast op basis van de uitkomst van een test/check.

Week 9-10-11
In week 9 is er volop gewerkt aan het Eindverslag en is ook het eerste commentaar van de SIG verwerkt. Daarnaast nog een aantal kleine bugs gefixt en gewerkt aan de Naieve Error Transformer, welke er voor moet zorgen dat er bekend is waarom bepaalde Error Pages niet goed ingeladen worden. Hiervoor wordt een XSD (Schema) gebruikt.
In de weken 10 en 11 is het eindverslag afgerond en herzien. Daarnaast is een begin gemaakt met de voorbereiding van de twee presentaties (1 bij Océ en 1 op de TU Delft).

Hoofdstuk 7 - Bevindingen

7.1 Error Message Page XML

Tijdens ons project hebben we het meest naar de XML bron van Error Message pagina's gekeken. Om de XML goed te kunnen parsen moet er een bepaalde structuur aangehouden worden in de bron. De parser (in dit geval een XSL transformatie) kan dan handig van deze structuur gebruik maken om je output in de vorm te gieten die wij willen hebben. In deze paragraaf komen onze bevindingen te staan over het transformeren en valideren van de Error Message pagina's.

Wij zijn ons er van bewust dat in onze aanbevelingen er voor een herstructurering van de error message pages gepleit wordt.Deze bevindingen zullen dienen als argumentatie voor de herstructurering. Indien deze argumenten niet overtuigend genoeg zijn, kan het op z'n minst als een gedetailleerde opsomming van verbeteringen gezien worden aan de huidige error message pages.

In de klasse ErrorPageParserTest worden alle Error Message Pages getest op hun 'parse-baarheid'.
Het eerste deel van de test is het transformeren van elke Error message page naar een Error Code Page. Daarna worden elk van de gecreëerde pagina's 'ge-validate' door middel van XML Schema. Als er een exception optreedt wordt dit gelogd in een errorLog. Exceptions kunnen optreden doordat de XSL transformatie faalt, of doordat de pagina's niet voldoen aan ons XML Schema.
Elke error message page die een exception opgooit wordt in een lijst gestopt met andere pages die op het zelfde punt faalden. Er zijn vier verschillende punten waar een exception opgegooid kan worden, dus zijn er 4 verschillende lijsten. Deze zijn:

- *Error message page transform failed list.*
- *Error code page validation failed list.*
- *Part page validation failed list.*
- *Setting page validation failed list.*

Daarnaast is er nog een 5de lijst met error message pages die goed door de test heen komen. De zogenaamde *'successList'*.

Om aan te geven hoe onze Error Page parser presteert volgt nu een korte samenvatting van de test:

Failed: 178
Failed on transform: 35
Failed on error validation: 0
Failed on part validation: 125
Failed on setting validation: 18
Parsed Succesfully: 548
Amount parsed: 726

Oftewel, 35 pagina's kunnen niet weergegeven worden in onze TSM, daarnaast zijn er 143 pagina's waarvan 1 of meer suspected parts & settings rijen niet goed weergegeven worden. Daarnaast zijn er 548 pagina's waar geen error is voorgekomen bij de transformatie, deze zullen ook zonder problemen ingeladen kunnen worden. Daarbij moet wel gezegd worden dat er een verschil is tussen het zonder problemen inladen van een pagina en het *correct* inladen van een pagina. Tussen de pagina's die correct ingeladen worden is er zo'n grote verscheidenheid aan manieren waarop informatie gepresenteerd kan worden, dat het ondoenlijk is om deze allemaal correct weer te laten geven.

Er zijn verschillende redenen waarom de test faalt op sommige error pages. Een hele hoop verschillende exceptions/errors komen voor in onze errorLog, maar de oorzaken hiervan zijn (uitputtend) als volgt:
Error Page Transform exceptions/errors:

1. Probeert meerdere Part/Setting pages aan te maken met als naam '' (een lege string)

2. Probeert een Part/Setting page aan te maken met in de naam een ‘/’, ‘<’, ‘>’, ‘ “ ‘ of een ‘ : ‘ (illegal characters)
3. Probeert een Part/Setting page aan te maken wanneer deze leeg is.

Error 1 - Probeert meerdere Part/Setting pages aan te maken met als naam ‘’ (een lege string) &
Error 2 - Probeert een Part/Setting page aan te maken met in de naam een ‘/’, ‘<’, ‘>’, ‘ “ ‘ of een ‘ : ‘ (illegal characters):
Error 1 & 2 komen voor omdat ons systeem vraagt om aparte XML files voor parts en settings. Aan deze files moeten namen gegeven worden, en het gaat mis bij het bepalen van deze namen. In principe zou je een unieke identifier (een nummer) willen hebben voor elke part en setting, zodat deze unieke identifier gebruikt kan worden als naam voor de XML files. Op het moment zit er in het TSM niks zoals dit, en hebben wij ‘the next best thing’ moeten gebruiken. Er is een kolom in de suspected parts & settings tabel genaamd ‘Description’. Hierin staat vaak een simpele naam als beschrijving van de suspect part of setting, bijvoorbeeld ‘paper wrinkled sensor’.
Het gaat fout wanneer in deze description bepaalde tekens voorkomen die ‘illegaal’ zijn in filenames: ‘/’, ‘<’, ‘>’, ‘ “ ‘ , etc. Ook kan het voorkomen dat de description van een bepaalde suspect leeg is. In eerste instantie is dit geen probleem, want ‘.xml’ is gewoon een goede filename. Je krijgt alleen problemen zodra er meerdere suspects zijn met een lege description, want XSL kan niet meerdere verschillende files outputten met dezelfde naam.

Part/Setting validation exceptions/errors:
1. Een bestelnummer van een part kan niet uitgelezen worden
2. Een indexnummer van een part is een string, bijvoorbeeld ‘HD’, ‘1L’ of ‘mem’
3. Een SDStest element zonder link naar een SDStest.

Error 1 - Een bestelnummer van een part kan niet uitgelezen worden:
Error 1 is van een lastigere aard. Er zijn een aantal verschillende mogelijkheden die voor deze error kunnen zorgen. De eerste mogelijkheid is dat een bepaalde suspect foutief aangemerkt wordt als part. In de huidige TSM wordt een *suspect* gedefinieerd in een <row> van een suspected parts & settings tabel, zoals in het voorbeeld XML hieronder. Voor ons systeem was het nodig om expliciet aan te geven wat parts zijn en wat settings zijn, omdat het gedrag anders is voor parts en settings. Om dit onderscheid te maken hebben wij gezegd: als er in de <entry> node een <ptxt> node zit die een <xref> node heeft, dan is het een part. Dit betekent eigenlijk dat er een link is naar een TSM PartsList, waardoor het mogelijk moet zijn om het bestelnummer van het onderdeel te vinden.

```
   <row>

<!-- Entry met colname=col1 bevat de link naar de Partslist-->
    <entry align="left" valign="top" colname="col1" morerows="0" rotate="0">
      <ptxt>
        <xref href="M1024" optional="0">Z-2048-16</xref>
      </ptxt>
    </entry>

<!-- Entry met colname=col2 bevat de naam van de part/setting-->
    <entry align="left" valign="top" colname="col2" morerows="0" rotate="0">
      <ptxt>Paper wrinkled sensor</ptxt>
    </entry>

<!-- Entry met colname=col3 bevat de links naar de SDStests-->
    <entry align="left" valign="top" colname="col3" morerows="0" rotate="0">
      <ptxt>
        <xref href="M8196" optional="0">SDS 13 3 007</xref>
      </ptxt>
    </entry>

<!-- Entry met colname=col4 bevat de "action" lijst -->
    <entry align="left" valign="top" colname="col4" morerows="0" rotate="0">
      <ptxt/>
    </entry>
```

```
<!-- Entry met colname=col7 bevat de links naar electrical Diagrams -->
   <entry align="left" valign="top" colname="col7" morerows="0" rotate="0">
     <ptxt>
       <xref href="M16392" optional="0">Wiring 8D16</xref>
     </ptxt>
   </entry>
 </row>
```

De text binnen het <xref> element van de eerste <entry> wordt altijd geschreven in de vorm Z-(nummer)-(indexNr), of Z(nummer)-(indexNr) (dus zonder het eerste ' - '). De TSM partslist is eigenlijk een tabel met een aantal rijen die elk gemerkt zijn met een index nummer. Om het bestelnummer van een part te vinden is het dus zaak om het indexNr '16' (in dit geval) te matchen met een 'posNo', zoals te zien is in het stukje voorbeeld XML hieronder.

```
<!-- een voorbeeld van een <item> node in een partslist, zoals altijd gevuld met nep-informatie-->
<item posno="16" quantity="1">

        <!-- het bestelnummer van het onderdeel -->
        <partno>32768</partno>
        <description>-SWITCH</description>

        <remark>
                <ptxt>16ZD, 17ZD, 18ZD, 19ZD</ptxt>
        </remark>
</item>
```

Wat kan er hier foutgaan?
Het Z-nummer-index text blokje kan niet goed geparsed worden. Bijvoorbeeld: Er wordt soms een dubbele index aangegeven in de vorm van Z-2048-16,32. Voor een mens leest dit makkelijk als index = 16 of 32, maar voor een computer is dat niet zo duidelijk. Daarnaast wordt er af en toe een index meegegeven die niet terug te vinden is in de TSM partslist, dus kan er geen bestelnummer gevonden worden.

Error 2 - Een indexnummer van een part is een string, bijvoorbeeld 'HD', '1L' of 'mem':

Error 2 is eigenlijk een limitatie die wij op hebben gezet in ons XML Schema. Voor ons is een index *altijd* een integer. Als het een string als 'HD', '1L' of 'mem' moet zijn, moet het geen index genoemd worden.

Error 3 - Een SDStest element zonder link naar een SDStest:
In het suspected parts & settings tabel is er een kolom genaamd 'Test'. Hierin staan over het algemeen links naar SDS-tests, in <entry> met als colname=col3. De reden waarom deze error voorkomt is dat er af en toe een stukje text (zonder link) in de <entry> node staat. Onze XSLT transformeert dit dan naar een <SDStest> element die niet voldoet aan de eisen zoals gesteld door ons XML Schema. Namelijk, dat het een link (<xref>) moet bevatten naar een SDS test.

7.2 Testen op correctheid van pagina's

In het tweede deel wordt getest op 'correctheid'. Oftewel, kunnen de getransformeerde Error Message pages zonder exceptions/errors ingeladen en naar de GUI overgezet worden en laat het alle informatie zien die aanwezig is?

Om het eerste gedeelte van deze vraag te checken breiden we onze ErrorPageParserTest code uit. Nu wordt er voor elke error page in de 'successList' van de vorige paragraaf de transformatie naar GUI elementen gemaakt. Om daarna de errors af te vangen die voorkomen bij het transformeren.

De eerste test run leverde ons deze resultaten op:

Amount of pages transformed: 548
Successfully transformed: 500
Unsuccessfully transformed: 48

Daarnaast printen wij voor elke niet-succesvolle transformatie de error/exception die opgegooid werd, samen met de error message page waar het bij voor komt. Hieruit bleek dat er 2 soorten errors uit de transformatie naar voren kwamen. Daarnaast werden er verschillende exceptions geprint buiten onze testcode om, dit waren 2 verschillende exceptions. De reden van de errors waren als volgt:

1. *Part/setting pages zonder naam werden fout afgehandeld in de code, er werd geen rekening mee gehouden.*
2. *Error Message Pages zonder Suspected Parts & Settings block werden verkeerd afgehandeld.*

Deze twee bugs in de code waren makkelijk te fixen en sindsdien geeft deze testcode een score terug als volgt:

Amount of pages transformed: 548
Successfully transformed: 548
Unsuccessfully transformed: 0

Helaas betekent dit niet dat elke pagina correct wordt weergegeven. Ons systeem krijgt in zoveel vormen informatie aangeboden, dat het in het huidige systeem onmogelijk is om deze allemaal correct weer te geven. Een automatische test op de correctheid van de pagina's die we binnen ons systeem laten zien hebben we alleen niet kunnen maken. Het kwantificeren van de kwaliteit van informatie op een pagina is een lastig probleem, zelfs als je het vergelijkt met een andere pagina, bijvoorbeeld dezelfde pagina maar in een ander TSM systeem.

Hoofdstuk 8 - Aanbevelingen

Tijdens het project zijn een aantal dingen naar boven gekomen rond het TSM waarvan wij dachten dat er verbeteringen voor mogelijk waren. Een aantal van deze zaken waren al bekend bij onder andere het ITC en daarmee zijn we aan de slag gegaan. Aangezien niet alles te verwerken was (of dat er een overhaul van TSM-XML bestanden nodig is) hebben we binnen de 2 maanden van het project een deel van ons systeem geïmplementeerd. Over het algemeen bevelen wij aan om te kijken naar ons ontwerp (te zien in hoofdstuk 5) en aan te passen aan de benodigdheden van het 'echte' TSM. Een paar aanbevelingen verdienen volgens ons om extra in de schijnwerpers gezet te worden. Hieronder volgt een korte samenvatting van die aanbevelingen in volgorde waarin ze genoemd worden (dus niet van belangrijkheid). Waarom we deze aanbevelingen maken leggen we uit in de rest van het hoofdstuk.

- Gebruik XML waar XML voor bedoelt is, dus geen lay-out technische zaken opslaan in XML.
- Alle pagina's uit de TSM moeten voldoen aan een vastgesteld format (bijvoorbeeld een XML Schema) en daar moet niet van afgeweken worden.
- Test & Check block link leggen met parts & settings.
- Elk Test & Check een unieke 'key'
- Test & Checks volgorde aanpassen op basis van statistiek
- Part Replaced / Setting Changed button extra functionaliteit geven.
- Voor elke part een eigen TSM pagina
- unieke ID voor parts.
- Computer leesbare link tussen parts en hun dis-assembly page.
- Een part voorzien van een basis beschrijving (bijvoorbeeld functionaliteit).
- Voor elke setting een eigen TSM pagina
- unieke ID voor settings.
- Link tussen SDS en TSM
- Automatische check op correctheid van pagina's

8.1 Error Code Page

Error code pages hebben wij het meest bestudeerd tijdens onze stage en hier hebben we ook een aantal aanbevelingen voor. De meest belangrijke is: Gebruik XML waar XML voor bedoeld is. Veel problemen die wij hadden, en nog steeds hebben, met het parsen van de Error Message Pages komen voort uit het definiëren van lay-out technische zaken in de bron XML, vooral de Suspected Parts & Settings tabellen. Wij bevelen daarom aan om Error Message pages op te zetten zoals onze error code pages opgezet zijn (te zien in hoofdstuk 5). Voor deze nieuwe pages valt dan makkelijk een XML Schema (een soort template) te schrijven, waardoor je er zeker van kan zijn dat een systeem *correct* een pagina weergeeft, zelfs als er pagina's dynamisch aangemaakt worden.

Test & Checks block

Met het Test & Check block kan ook nog veel gedaan worden. Op het moment zit er in TSM geen link tussen Suspected Parts & Settings en het Test & Check block. Het toevoegen van expliciete links vanuit een Test naar een suspect heeft de volgende voordelen: Volgorde van suspects kan aangepast worden aan de hand van feedback over een uitgevoerde test, wat al wel geimplementeerd is in DynamicTSM. Als er voor een bepaalde test geen link naar een suspect is, waarom bestaat deze test dan? En andersom, als er vanuit een test een 'link' is naar een suspect die in het Suspected Parts & Settings tabel staat, waarom staat deze er niet in? In principe is suspect een allesomvattende term, elke mogelijk oorzaak van een error is een suspect. Natuurlijk Parts & Settings, maar ook oorzaken als "het papier zit niet goed in de lade".

De volgorde van tests kan aangepast worden aan de statistiek die beschikbaar is over parts waar de test 'over gaat'. De test met gemiddeld of gezamenlijk de hoogste kans op het geven van uitsluitsel komen dan op een hogere positie in de lijst. Als er op een Test Succeeded' of 'Test Failed' button gedrukt wordt, kan hiervan een notitie gemaakt worden in een database. Deze statistiek zou daarna ook meegenomen kunnen worden in de bepaling van de positie van een test in de lijst.

Op het moment zit dit nog niet in het DynamicTSM systeem. Je zou hiervoor de klasse TSMTestCheckBlockPanel.java aan moeten passen, zodanig dat er statistiek over parts opgevraagd kan worden aan de StatisticsManager klasse. Op basis van deze statistiek is het dan mogelijk om de Test & Checks te sorteren.
Om een te kunnen notitie te kunnen maken van het succes of falen van een Test/Check moeten er unieke keys gegeven worden aan Tests & Checks, anders krijg je problemen van hetzelfde caliber als beschreven in hoofdstuk 7. Hier bedoelen we specifiek het refereren naar een Test of Check.

Suspected Parts & Settings block
In het DynamicTSM is er voor elke part en setting een 'part replaced' of 'setting changed' button te vinden. Op het moment maakt het systeem, als de gebruiker op 1 van deze buttons klikt, een notitie in tekst-bestand waarmee aangegeven wordt dat het onderdeel is vervangen of de setting verandert is. Uiteindelijk is het de bedoeling dat er een entry in de datalog[3] van de printer komt. Naast een entry in de datalog kan er nog meer functionaliteit aan de 'part replaced' buttons gehangen worden. Er zou automatisch een notitie gemaakt kunnen worden in een soort van 'inventory', waar software geschreven kan worden zodat de monteur aan het einde van een service call minder administratieve handelingen hoeft uit te voeren.

Om extra functionaliteit aan de 'part replaced' button te hangen, moet er in de TSMSuspectBlockPanel klasse de handleSetting / handlePart methode aangepast worden. Daarnaast bestaat er nog niks van inventory binnen het systeem, maar er zou heel makkelijk een InventoryManager toegevoegd kunnen worden. Deze InventoryManager zou dan notities kunnen maken van vervangen onderdelen op, bijvoorbeeld, een memory-stick, waarna er software voor de laptop geschreven moet worden die deze notities uit kan lezen. Deze zelfde software zou dan op de memory-stick kunnen zetten welke onderdelen in de 'inventory' zitten en welke de monteur bij zou moeten bestellen, waarna het DynamicTSM systeem dit weer kan gebruiken in de part pages (die hieronder genoemd worden).

8.2 Part Page

Alle informatie over onderdelen wordt in het huidige systeem verspreid opgeslagen over 3 verschillende soorten pagina's (Error Message, dis-assembly en partslist pages). De meeste informatie staat op de error message pagina's, zelfs zoveel dat een part eigenlijk 'bestaat' in deze page. Het is daarom ook niet mogelijk om los van een error message page een part page op te vragen. Dit is, daarentegen, wel nodig als er een Preventive Maintenance lijst (zoals gedefinieerd in paragraaf 5.4) gemaakt zou worden.
Om, vanuit de Preventive Maintenance lijst te kunnen verwijzen naar deze part pages is er een unieke 'key' nodig die specifiek is voor een onderdeel. Zoals in hoofdstuk 7 is beschreven komt het gros van de problemen met het parsen doordat er geen goede naam voor de part page gekozen kan worden. Dit zou in ieder geval ons systeem een stuk soepeler laten draaien. De part page (en zijn XML) zou er dan zo uit kunnen zien als dat wij beschreven hebben in hoofdstuk 5. Daarnaast bevordert het hergebruik binnen het TSM en als een part aangepast wordt (bijvoorbeeld de naam verandert) dan wordt dit gelijk weergegeven in alle pagina's die naar deze part verwijzen.

[3] Datalog - Een Database met veel informatie over o.a. veranderingen aan de printer.

Part pages bestaan in ons systeem al, maar de description en dis-assembly pages zijn nog niet goed geïntegreerd, mede omdat hier nog geen XSLT voor is gebouwd. Mogelijke problemen met het transformeren van deze dis-assembly pages zijn ook nog niet aan het licht gekomen. Om de weergave van part pages in DynamicTSM aan te passen moet de klasse TSMPartPagePanel aangepast worden, zodanig dat bijvoorbeeld de description en de dis-assembly goed uit- en ingelezen kan worden.
Voor zowel de part en setting 'key' kan er binnen de bijbehorende <row>/<entry> een nieuw veld gedefinieerd worden die deze key bevat. Dit zou niet optimaal zijn, maar het is dan in ieder geval mogelijk om, vanuit ons systeem, statistiek op te slaan over deze suspects.

Link tussen part en dis-assembly page

Een belangrijk onderdeel van Preventive Maintenance is het daadwerkelijke vervangen van de onderdelen. Dit staat allemaal, voor zover wij kunnen beoordelen, goed in het TSM beschreven. Het enige dat mist is de expliciete link tussen deze vervangings pagina's, genaamd dis-assembly pages, en de onderdelen van de printer die vervangen worden. De enige manier om deze link tussen de pagina's te leggen is het uitlezen van een error message page en hopen dat er van daaruit naar een dis-assembly page gelinkt wordt. Dit maakt het totaal onmogelijk om vanuit een part page de dis-assembly pagina die erbij hoort te vinden.
Wij stellen daarom voor: Voeg aan elke dis-assembly page een lijstje toe met onderdelen die met behulp van de instructies die op die pagina staan vervangen kunnen worden, oftewel een lijstje met partID's. Daarnaast voeg je aan elke part page een expliciete link toe naar de dis-assembly page, indien dat bestaat voor het onderdeel.

8.4 Setting Page

Om consistentie te blijven houden bevelen wij ook aan om voor settings aparte pagina's te maken. Setting pages staan niet in ons design doc, omdat we hier te laat mee begonnen zijn, maar er zijn zeker voordelen aan het hebben van een aparte pagina voor elke setting. Wat op een setting page zou komen is een history log over wanneer, hoe vaak en met welke waarde een bepaalde setting is aangepast. Dit soort informatie geeft monteurs vanuit het TSM de kans om snel te bekijken of er met deze setting iets raars is gebeurt. Daarnaast zouden de SDS Tests in het scherm opgenomen kunnen worden, zodat een monteur tegelijk de history log kan bekijken en de waarde van de setting veranderen.
Om deze setting page te kunnen maken, is er ook een unieke key voor elke setting nodig, zodat wederom andere pagina's naar settings kunnen verwijzen.

Setting pages zitten nog niet in het DynamicTSM systeem. Een guiElement dat deze beschrijft zou toegevoegd moeten worden. Er is wel al een SettingManager aanwezig die een setting pagina uit leest en hem retourneert, er moet hier alleen de conversie naar een SettingPage gemaakt worden, die nog niet bestaat.

8.5 Uitbreidingen op software

Link SDS met DynamicTSM
Om uiteindelijk de link te maken tussen het SDS en het TSM moet er vanuit het SDS, zodra er op een link geklikt wordt, een aanvraag gedaan worden naar het DynamicTSM om een bepaalde pagina in te laden. Op het moment is de
ze link er nog niet. Om dit toe te kunnen voegen moet er binnen de SDS code onze ViewManager aangeroepen worden, specifiek de loadInOverviewWindow() methode. Deze verwacht een string van de vorm (pageType)/(pageName) (error/11504 of part/Paper Tray Motor) en retourneert een Panel met de juiste pagina.

Implementatie van de 4 andere entry-points
In hoofdstuk 5 staat beschreven wat onze ideeën zijn bij de andere 4 entry-points. Om DynamicTSM compleet te maken zouden de pagina's die wij bespreken ook toegevoegd moeten worden. Hier hebben wij ruwe concepten op papier gezet en hebben we met o.a. Part Pages rekening gehouden met deze eventuele uitbreidingen.

8.6 Miscellaneous

Automatische check op correctheid van TSM pagina's (in vergelijking tot andere)

Om te zorgen dat de overgang van het Lotus Notes TSM naar het DynamicTSM (of een ander TSM systeem) zo soepel mogelijk verloopt, is het belangrijk dat pagina's goed weergegeven kunnen worden in het DynamicTSM. Handmatig elke pagina nalopen of deze klopt is een erg tijdsintensieve taak, dus zou het handig zijn als een programma dit voor je zou kunnen checken. Op het moment is het onmogelijk om het programma te laten oordelen over de kwaliteit van informatie, laat staan deze te corrigeren, maar er zijn wel bepaalde metrieken die gebruikt kunnen worden.
Voor het automatisch checken van code zijn al veel technieken uitgevonden en een aantal hiervan zouden wellicht toepasbaar kunnen zijn om de kwaliteit van 2 TSM's tegen elkaar uit te meten. Bepaalde pagina's die bij bepaalde metrieken (bv, word count, link count, working link count) bepaalde thresholds overschrijden kunnen dan onder de aandacht gebracht worden van de gebruiker om te kijken wat daar mis gaat.
Het overzetten van het TSM naar een totaal nieuw systeem zal altijd met de nodige problemen komen, dit zou het makkelijker moeten maken om de oorzaak van sommige problemen te lokaliseren.

Hoofdstuk 9 - Evaluatie

In dit hoofdstuk worden een tweetal soorten evaluaties gegeven. Ten eerste (9.1) een algemene evaluatie van hoe wij het project ervaren hebben en daarna een code evaluatie zoals deze gedaan is vanuit de Software Improvement Group (SIG), verbonden aan de TU Delft. In appendix E staat het commentaar van de 1e en 2e keer en in 9.2 zullen wij reageren op hoe wij hier mee om zijn gegaan.

9.1 Algemene Evaluatie

Waarschijnlijk hebben we ons een beetje verkeken op de omvang van het hele project. De vrijheid die we gekregen hebben speelt hierbij een rol, aangezien de opdracht vrij algemeen geformuleerd is.
Mede daardoor is er een design gemaakt voor een (praktisch) volledige herstructurering van de huidige TSM, terwijl er tijdens de implementatie slechts een beperkt deel hiervan daadwerkelijk opgezet is (De Error Page en Part Page). Er is veel tijd gestoken (een maand) in het vergaren van alle informatie die nodig was en om de designs op te stellen.

Om het volledige systeem te implementeren zoals het in het design is opgenomen, zou het noodzakelijk zijn om nog enkele maanden aan het project vast te plakken, wat uiteraard niet mogelijk is.

De samenwerking is in wezen heel goed gegaan. Aangezien we een groep van slechts 2 studenten hadden waren er geen vergaderingen nodig (al was er elke ochtend een Stand-Up Scrum met de naaste collega's) en het feit dat we een eigen werkplek hadden en ruimten konden reserveren voor designing maakte het werken prettig en makkelijk.

Als we het project opnieuw zouden doen, zouden we waarschijnlijk het merendeel op dezelfde manier te doen, al zouden we het eerder (en beter) afgebakend hebben. Aan de ene kant ligt er nu een ontwerp voor een volledig systeem. Aan de andere kant hebben we (te) weinig rekening gehouden met het feit dat het veel meer tijd zou gaan kosten om alles te implementeren. Hierdoor hebben we een risico genomen dat we aan het eind misschien niet een volledig prototype af gehad zouden hebben.

Verder is er tijdens de implementatiefase onbewust voor gekozen om niet uitgebreid dingen te testen (met bijvoorbeeld JUnit Testing). Het enige dat op grote schaal getest is, is het inlezen en parsen van de TSM-files, aangezien dit veel problemen opleverde. Bugfixing en error handling werd tijdens het programmeren met trial-and-error gedaan en dingen werden aangepast zodra er een fout optrad.

Mede dankzij de prima werkomgeving en de zeer prettige wijze waarop betrokken collega's ons te woord stonden is dit project een hele leuke en goede ervaring geweest.

9.2 Code Evaluatie

De code is tweemaal gecontroleerd op onderhoudbaarheid door SIG en daar is commentaar op teruggekomen. Tussen de 2 evaluaties (29 augustus en 15 september) is het eerste commentaar verwerkt en zijn er nog wat delen toegevoegd aan de code die bij de 2e evaluatie meegenomen zijn.

9.2.1 Eerste evaluatie: 29 augustus 2011

De eerste evaluatie was zeer positief en het commentaar kwam voornamelijk neer op dat een aantal methoden binnen de code te lang waren (zoals in een testklasse). Waar mogelijk zijn deze code-blocks kleiner gemaakt door middel van refactoring. Verder is er op gelet bij nieuwe methoden dat daar niet dezelfde fouten zouden optreden.

Daarnaast was er binnen het project een package genaamd “JTestUnits” toegevoegd, met als intentie om hier Unit Tests in te stoppen. Echter bleek het systeem niet geschikt te zijn hiervoor, omdat een groot gedeelte van het project bestaat uit het inlezen en parsen van TSM bestanden en deze blokken weer te geven op de GUI. Er is daarom besloten om deze package te verwijderen. Daar voor in de plaats wordt de parser wel uitgebreid getest door middel van XML Schema’s.

9.2.2 Tweede evaluatie: 15 september 2011

De tweede evaluatie was zo mogelijk nog positiever dan de eerste evaluatie. Het commentaar dat gegeven was op complexe en lange methodes is overal verwerkt.

Het advies om de testklasse rondom de XML validatie ook aan te pakken hebben we naast ons neer gelegd. Het doel van deze klasse was om zoveel mogelijk informatie over het parsen te verzamelen per error code page die geparsed werd. Zoals in hoofdstuk 7 uitgelegd word, moeten de te verzamelen stukjes informatie in aparte lijsten opgeslagen worden, als het allemaal op 1 hoop komt heb je er niks aan. Deze verschillende lijsten zijn eigenlijk allemaal output van de test functie. Er zijn 2 loops (voor de setting en voor de parts) die redelijk op elkaar lijken, maar beide in verschillende lijsten hun output stoppen en daarnaast ook nog 3 verschillende variabelen aanpassen. Het refactoren van deze code zou een methode opleveren waar 3 output variabelen voor nodig zijn. Wij beschouwen dit als nog minder goed onderhoudbare code dan de huidige loops, dus hebben we het niet aangepast.

Hoofdstuk 10 - Conclusie

Het doel van deze opdracht was onderzoek doen naar de huidige Service Software (STAR, SDS en de bijbehorende TSM). Aangezien dit een erg breed onderwerp is, is er gekozen om ons volledig te richten op het gedeelte rond de TSM.

Hiervoor is uitgebreid onderzoek gedaan naar de TSM en door middel van zelf rondkijken, veel vragen stellen en gesprekken voeren met betrokkenen is er een beeld ontstaan over de huidige staat van dit systeem.

Aangezien er een beperkte tijd was om het systeem te implementeren is slechts een deel van het volledige design geïmplementeerd. Er is gekozen om het gedeelte rond de Error Codes uit te werken, aangezien dit, samen met diagnose d.m.v. symptomen, een redelijk groot deel van het gebruik van de TSM vormt. Het resultaat hiervan is een GUI waarop Error Pagina's goed (op een enkele uitzondering na) ingeladen kunnen worden en dynamisch aangepast worden wanneer dit nodig is aan de hand van Tests & Checks en algemene Statistieken wanneer die beschikbaar zijn. Daarnaast is een basisimplementatie voor de Parts Page aanwezig, die als voorbeeld kan dienen voor een soortgelijke Settings Page.

Er is gebleken dat een gedeelte van het huidige TSM systeem gewoon hergebruikt kan worden, echter volgt uit onze aanbevelingen dat er zeker een aantal dingen aangepast en / of verbeterd kunnen worden, vooral aan de XML. Het streven om af te stappen van Lotus Notes is een van de dingen die absoluut haalbaar zou moeten zijn.

De overige delen kunnen in een later stadium toegevoegd worden aan ons systeem. Het framework is opgezet op een manier dat dit geen problemen op zou moeten leveren. Wij zien daarom het huidige DynamicTSM systeem als een goede eerste stap in de volledige herstructurering van het TSM, zodat het gebruik kan maken van de mogelijkheden die de printer biedt.

Dan rest er alleen nog de vraag; Waarom zou Océ tijd en resources vrij willen maken om verder onderzoek richting het DynamicTSM uit te voeren?

Het antwoord hierop is: Efficiëntie. Elke minuut die een monteur aan zijn software besteedt die niet rechtstreeks met het oplossen van printer problemen te maken heeft is er 1 te veel. Typefouten tijdens het zoeken naar TSM pagina's, het doen van tests die een kleine kans op slagen hebben, parts opzoeken die een kleine kans van falen hebben kosten allemaal tijd. Oftewel, met het overbrengen van het TSM naar de printer is er veel tijdswinst tussen service calls te halen en het uitwerken van de ideeën voor het DynamicTSM zou hier een perfecte eerste stap voor zijn.

Hoofdstuk 11. Bronnen

1. Brunner, M. 2009. Analysis Figures from the Field - Internal Memo.
2. Tidwell, D. 2008. XSLT, Mastering XML Transformations. 2nd ed. O'Reilly. 985 p.
3. Tunnissen, H. 2011. HTML TSM Requirements Documentatie.
4. Vlist, E. van der. 2002. XML Schema. 1st ed. O'Reilly. 400 p.
5. Wang, L. 2007. Internal internship report.

Appendix A - Oriëntatie verslag

Stage bij Océ

Gemaakt door:
Erwin Haasnoot
Jos Kuijpers

Begeleider Océ:
Marvin Brunner

Begeleider TU Delft:
Peter van Nieuwenhuizen

Inhoudsopgave

Voorwoord

Dit oriëntatieverslag bevat de informatie die we tijdens de oriëntatiefase voor het Bachelor-project T*ouch screen user interface design for Service* tot ons hebben genomen. Dit project is uiteindelijk veranderd in een algemener *improve service software* project. Het project vindt plaats in het Software Engineering 1 departement van de Research & Development afdeling van Océ. In deze fase zijn er voornamelijk gesprekken gevoerd met betrokken personen en instanties. Hiervan zijn korte verslagen / samenvattingen terug te vinden in dit document.

Introductie

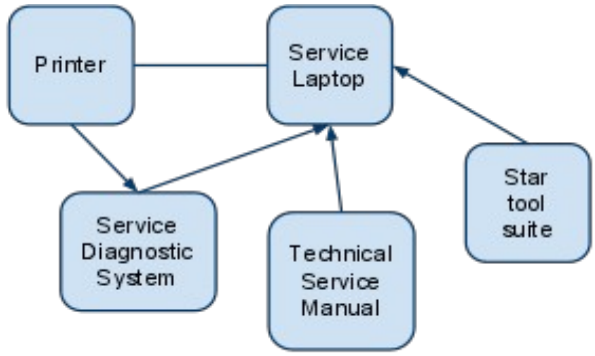
Voor het oplossen van problemen die optreden bij de printers van Océ wordt gebruik gemaakt van een combinatie van een Technical Service Manual (TSM) en een Service Diagnostic System (SDS). Het SDS systeem is momenteel in overgang van een active-x applicatie naar een web-based systeem. Het TSM echter hangt nog steeds aan Lotus Notes, een database systeem van IBM, vast en dat beperkt de portabiliteit van het systeem aanzienlijk. Binnen ons project zullen wij uitzoeken wat er aan de software van service verbeterd kan worden en hoe wij daar aan bij kunnen dragen.
Om een zinvolle bijdrage te kunnen leveren is het belangrijk dat we ons oriënteren op het project. Wij hebben daarom in de oriëntatieweek een aantal gesprekken, gepland en ongepland, gehad. Hiervan hebben wij kort opgeschreven welke informatie we eruit hebben gehaald. Na een aantal gesprekken wisten wij zeker dat we ons toe wilden spitsen op het TSM, dus hebben we de verdere gesprekken zo gepland dat we zoveel mogelijk hierover te weten konden komen. De structuur van het TSM wordt daarom uitvoerig in dit verslag besproken, alsmede de stukken waar onze kennis van het TSM nog niet voldoende is. Daarnaast zullen wij, voor zover dat mogelijk is, de tools die we gebruiken bespreken en de reden waarom we hiervoor gekozen hebben.
In hoofdstuk 1 vallen de verslagen van de gesprekken die we gevoerd hebben terug te lezen. In hoofdstuk 2 wordt er uitgelegd hoe het TSM in elkaar zit.

Hoofdstuk 1 - Verslagen van gesprekken

Gesprek Marvin Brunner over project - 4 juli

Onze eerste dag bij Océ was gevuld met kennismakingsgesprekken en oriënterende gesprekken. Het gesprek dat we met Marvin Brunner hebben gehad was een van de belangrijkere gesprekken van die dag. We hebben hier uitleg gekregen over het doel van ons project.

Ons project heeft als doel het verbeteren van de manier waarop service-monteurs hun taak uit kunnen voeren. Hoe we dat precies gingen doen was aan ons. Om een idee te krijgen van hoe service in elkaar steekt hebben we uitleg gekregen over de software pakketten die gebruikt worden en over de problemen die hiermee ontstaan.
Het deel van de service-software waar wij ons op moeten focussen zit als volgt in elkaar; Een service laptop wordt op de printer aangesloten. Deze laptop bevat een software suite genaamd Star tools, alsmede een documentatie database genaamd het Technical Service Manual (TSM). Daarnaast wordt op de printer een web-applicatie genaamd het Service Diagnostic System (SDS) gehost, die op de Service Laptop gedraaid kan worden (in een browser).



Figuur 1.1. Schematische weergave van een deel van het service softwarepakket

Star tools is een software suite dat beheerd wordt door een software-groep binnen Océ HQ. Er zitten een aantal tools in deze software suite die regelmatig door service monteurs gebruikt worden om bijvoorbeeld bepaalde statistieken bij te houden over printers en op basis daarvan een werklijst[4] aan te maken voor service monteurs. Tussen Star tools en het TSM en SDS is geen koppeling. Ontwikkeling van Star tools ligt al een lange tijd stil en Research & Development (R&D) heeft geen rechten om dit te onderhouden.
Het TSM is een documenten database in Lotus Notes waar monteurs in kunnen zoeken om bepaalde informatie over printers op te halen. Denk hierbij aan; Hoe installeer je een bepaalde printer, wat moet je doen bij een bepaalde fout in de printer. De details van het TSM worden in hoofdstuk 2 besproken.
Het probleem met het TSM is dat het lastig/onhandig in gebruik is. Ervaren service monteurs gebruiken het bijna niet, zij vertrouwen meer op hun ervaring. Onervaren monteurs zijn veel tijd kwijt met het zoeken naar pagina's die ze nodig hebben, vanwege de onhandige interface (mede te danken aan het gebruik van Lotus Notes).
Als voorbeeld gaf Marvin de installatie documentatie voor printers. Printers installeren, fysiek maar ook software, is een complexe klus, en het staat tot in detail in het TSM beschreven. Alleen door de navigatie mogelijkheden in Lotus Notes is het vaak heel lastig bij te houden op welke stap van de installatie je precies zit. Er wordt niet aangegeven welke pagina's je al geopend hebt, en elke pagina die je opent, opent een nieuw venster binnen Lotus Notes, wat niet helpt bij het navigeren door het TSM. Daarnaast laat de portabiliteit van het TSM ook te wensen over. Door het gebruik van Lotus Notes is het alleen mogelijk om het TSM te bekijken op PC's, en dan ook alleen als je de (hoge) licentie kosten betaalt voor het gebruik van Lotus Notes. Concluderend zijn er een redelijk aantal verbeteringen mogelijk door het TSM over te zetten naar een web-applicatie.

Het SDS is een applicatie die recent onder handen is genomen, onder andere door Marvin Brunner zelf. Waar het eerst nog een tool was die op de service laptop draaide, is het nu een webapp die gehost wordt op de printer zelf. Het SDS geeft o.a. toegang tot statistieken (Error Logs, modification logs), configuratie opties, bepaalde tests uitvoeren (om te zien of de printer nog goed werkt) en de mogelijkheid om een back-up te maken van dit alles. Het ontbreekt in dit nieuwe SDS nog aan koppeling met het TSM en Star Tools. Alhoewel het in dit geval vanuit het SDS wel mogelijk is, zijn het de andere tools die tegenwerken.

[4] Werklijst - een lijst met onderdelen die mogelijk defect zijn of onderhoud nodig hebben. Aan de hand hiervan kan een monteur de printer repareren of onderhouden

Over het algemeen zijn de problemen met de collectie service-tools bij iedereen bekend, maar is er niet genoeg motivatie/geld beschikbaar om dit daadwerkelijk op te lossen. Volgens Marvin zou het werk dat wij verrichten mogelijk kunnen helpen bij het motiveren van de juiste mensen om de service tools over te zetten naar een systeem, zodanig dat integratie veel beter mogelijk wordt en veel van de zwakke punten van het systeem verdwijnen.

Conclusie:
De huidige structuur rond de Service bestaat uit een aantal losse applicaties (TSM, STAR Tools en SDS) en het combineren hiervan is een stap in de goede richting. Aangezien de complete integratie van deze 3 systemen niet haalbaar is binnen ons project maken wij de keuze om ons te richten op de TSM. (**Note:** Later in het project is deze keuze aangepast naar het volgende: het integreren van de TSM en het SDS-systeem op de printer zelf)

Gesprek Ruud Sampers over Web-SDS - 5 juli

De web-SDS client is een relatief recente ontwikkeling binnen R&D en is de opvolger van de oude SDS client die gebruik maakte van activeX componenten. Ruud heeft ons een demo gegeven over hoe deze web-SDS client in elkaar zit en wat je er allemaal mee kan doen.

Conclusie:
Aangezien de web-SDS client nog niet volledig operationeel is hebben we besloten hier verder niets mee te doen.

Gesprek Eddie Gunther korte demo TSM - 5 juli

Na de demo heeft Ruud voor ons nagevraagd of er een service laptop beschikbaar was om in rond te kunnen kijken. Op dezelfde verdieping staat een testopstelling waar op dat moment aan gewerkt werd. De bijbehorende service laptop was niet direct in gebruik en Eddie Gunther heeft ons een korte demo (+-30 minuten) gegeven over de werking van een paar delen van het TSM en het SDS gedeelte. Zo liet hij onder andere zien wat een Service Monteur kan proberen als er een bepaalde error code optreedt door simpelweg de zoekfunctie van Lotus Notes te gebruiken. Hierbij werd wel de gebruikelijke zoekboom (Table of Contents) genegeerd, welke in veel gevallen een betere optie is dan de zoekfunctie van Lotus Notes te gebruiken.

Het was beter geweest als we ook zelf dingen hadden kunnen uitproberen, maar aangezien er op dat moment aan de printer zelf gesleuteld werd was dit helaas niet mogelijk.

Note:
Later in die week hebben we toegang gekregen tot de TSM binnen Lotus Notes en was het niet meer nodig om op de service laptop dingen te checken, aangezien alles wat we nodig hadden beschikbaar kwam.

Gesprek met Ruud Sampers, over testen - 6 juli

Ruud had bij de stand-up meeting van 5 juli gemeld dat hij op 6 juli een aantal uren zou besteden aan tests en er was afgesproken dat we daar bij konden zijn om te observeren en om vragen te stellen. Bij deze tests zijn een aantal problemen op een eerder tijdstip vastgesteld en door iemand anders opgelost. De taak van de tester in dit verband was checken of het probleem inderdaad was opgelost (als een reviewer van het probleem). Wat in een aantal gevallen overigens niet het geval bleek te zijn. Zo kwam er een incomplete test voorbij en een test die ronduit verkeerd was.

Een groot gedeelte van de dag hebben we geobserveerd in de testruimte hoe er een aantal tests uitgevoerd werden en hebben we wat vragen kunnen stellen over het hele proces daar omheen. Later op de dag kregen we de mogelijkheid aangeboden om zelf een kleine test te doen die door onze begeleider was uitgevoerd en na wat zoekwerk is dit ook gelukt. Hierbij hadden we ook de mogelijkheid om wat beter in de SDS software rond te kijken die gebruikt wordt.

Conclusie:
Het testen van een printer is belangrijk, maar niet van belang voor ons project. Al was het wel een nuttige ervaring overall.

Gesprek met Marvin Brunner - 11 juli ('s middags)
's Ochtends is dhr. Nieuwenhuizen op bezoek geweest in Venlo, hier hebben wij vooral een aantal praktische zaken besproken (deliverables, deadlines, enz.). In navolging van dat gesprek hebben Erwin en Marvin (Jos was ziek naar huis) nog gepraat over een paar specifieke zaken en ook over het TSM. Uit dit gesprek kwam naar voren dat een aantal stukken functionaliteit van Lotus Notes niet werken voor het TSM, of zelfs het gebruiksgemak verminderen. Hierbij noemde hij specifiek het knipperen van bepaalde links op een pagina. Het is een functie die geïmplementeerd is om het mogelijk te maken het TSM te bedienen met een toetsenbord, maar in de praktijk leidt het constante knipperen heel erg af. Ook klikken veel mensen op de eerste de beste blauwe link, zonder dat er wordt gekeken naar de relevantie van deze link.

Conclusie:
De weergave van de pagina is van groot belang. Het moet overzichtelijk zijn en "to-the-point". Lotus Notes hangt extra functionaliteit aan een pagina (de knipperende link) en dat leidt alleen maar af. Daarnaast is Lotus Notes niet helemaal geschikt voor de TSM.

Gesprek met Pieter Dielissen (ITC) over TSM (met uitgebreide Demo) - 12 juli
De volgende belangrijke afspraak was gemaakt met een van de developers van de TSM, Pieter Dielissen. Hij heeft ons een zeer uitgebreide (bijna 2 ½ uur) demonstratie gegeven van de TSM. Daarbij is veel uitleg gegeven over hoe het nu *exact* in elkaar zit, welke keuzes er gemaakt zijn en hoe zaken als distributie geregeld zijn. Ook werd er antwoord gegeven op een aantal belangrijke vragen die we nog hadden rondom onder andere de opbouw van de TSM (zoals de volgorde waarin oplossingen gegeven worden).

Om te beginnen is de layout van het systeem beschreven, waarbij voornamelijk de aandacht uitging naar het TSM gedeelte, maar de andere onderdelen (bijvoorbeeld: Externe Documentatie: voor randapparatuur die niet van Océ is; Partslist: spreekt voor zich; enz) wel kort besproken zijn en de functie / keuzes daarvoor zijn uitgelegd. Zo is de Externe Documentatie een losse database die niet gecombineerd is met de TSM, aangezien hierin voornamelijk PDF bestanden staan en de size was toen een limiterende factor (2-56k "lijn") om over te sturen. Later zou het geen probleem meer zijn, maar is er voor gekozen de twee databases niet te integreren.

Om niet alles te herhalen binnen dit verslag, wordt in Hoofdstuk 2 de TSM verder en meer uitgebreid toegelicht.

Wel werden er een aantal Requirements gemeld die in de visie van (o.a.) Pieter Dielissen zeker in een nieuwe / verbeterde variant van de TSM (zouden) horen:

- Het annotatie systeem. (zie Hoofdstuk 2). Hierbij moet opgemerkt worden dat het merendeel van de openbare annotaties aangemaakt wordt door HQ zelf en niet door de Service Technici in het veld.
- Het gebruik van statistische data om veelvoorkomende problemen sneller te kunnen vinden in de TSM
- De database omvormen tot een database die alleen de M(odule)-nummers bevat uit de TSM en de bijbehorende pagina pas opbouwt als deze geopend wordt. Dit kan de grootte van de database drastisch verlagen.
- De TSM opslaan en uitvoeren via een web-applicatie op de printer zelf, zodat de servicelaptop niet per definitie meer nodig is.

Conclusie:
Uit het gesprek is de werking van de TSM en de opbouw hiervan via het Content Management Systeem duidelijker geworden. Dit is van belang voor het project aangezien onze TSM ook bestanden moet kunnen inlezen. De lay-out van de input is belangrijk om deze op de juiste manier te verwerken. Daarnaast zijn er een aantal wensen vermeld en we hebben gekozen om te kijken in hoeverre deze te verwerken zijn.

Meeloopdag met Frank-Jan van Beers (Service Technicus) - 14 juli
Ter verdere oriëntatie op het daadwerkelijke gebruik van de TSM is er op 14 juli een meeloopdag geregeld met een Service Technicus. Er was aan het begin van de dag exact 1 Service Call waar hij naartoe zou gaan, dus het was afwachten hoe lang het zou duren. Uiteindelijk bleken de problemen met de printer (een ouder model kleurenprinter van Océ dat niet meer geproduceerd wordt) de gehele dag in beslag te nemen. Bij aankomst bleek er die ochtend een nieuwe error opgetreden te zijn die volgens Frank-Jan regelmatig voorkomt bij dit type.

Hij hoefde de TSM dan ook niet te raadplegen en ging meteen sleutelen. Toen bleek dat na de vervanging van enkele beschadigde onderdelen het oorspronkelijke probleem (strepen op het papier die door die beschadigingen worden veroorzaakt) wel opgelost was, was het nieuwere probleem hiermee nog niet opgelost. Terug naar de TSM en na daar kort naar gekeken te hebben kwam de conclusie dat daarin de oplossing niet gevonden zou worden. Volgens Frank-Jan waren de onderdelen die net vervangen waren verantwoordelijk voor 8 van de 10 gevallen van de melding en dit werd niet genoemd binnen de TSM. De oplossing werd gevonden bij een onderdeel dat als gevolg van de eerder opgetreden error kapot was gegaan (een breekpin in de motor). Na deze vervangen te hebben werkte dat deel weer.

Het tweede gedeelte van de dag is besteed aan het fixen van het andere probleem, waarbij ook dit keer de TSM absoluut geen uitkomst bracht. Een setting binnen de printopties was door iemand uitgezet en daardoor was het probleem ontstaan. Setting teruggezet, paar onderdelen preventief vervangen en het werkte weer prima. Beide oplossingen werden niet genoemd binnen de TSM.

Tussendoor gaf Frank-Jan uitgebreide uitleg over het proces van de reparatie en beantwoorde hij mijn vragen. Daaruit bleek dat hij de TSM niet vaak gebruikte, maar dat hij wel opmerkingen had over het systeem. Deze kwamen onder andere voort uit ervaringen van andere Service Technici (het voorbeeld was een probleem waarbij 2 verschillende printplaten het probleem kunnen zijn. Vervangen kost veel tijd en de volgorde waarin ze staan bij de suggesties is gebaseerd op hoeveelheid tijd die het kost om een onderdeel te testen en te vervangen).

Ondanks dat de Technicus zelf de TSM niet heel vaak gebruikt, heeft hij toch feedback gegeven waar we wat mee kunnen, dus was het een nuttige dag geweest.

Conclusie:
Meer inzichten verkregen over het gebruik van de TSM en hoe de servicemonteur deze gebruikt. Hieruit hebben we de keuzes gemaakt over onder andere de inrichting van ons eigen systeem. Zo gaan we informatie over onderdelen beschikbaar maken op de Error Pagina al, zodat bijvoorbeeld het opzoeken van een Part-ID nummer (voor een bestelling) minder tijd kost.

Gesprek met Hans Tunnissen, Marvin Brunner en Rene Titulaer over Web-TSM - 15 juli
Binnen Océ is er recent een project gestart voor de ontwikkeling van een Web-based TSM systeem, opgezet vanuit HQ, die de opdracht had aangedragen. Het was van belang om een duidelijk beeld te krijgen met betrekking tot wat hun plannen waren, in welk stadium deze zich bevonden en of dit invloed zou hebben op ons eigen project.

De plannen zijn om alle huidige gewenste functionaliteiten te behouden, waarbij ook gekeken wordt naar integratie van de SDS en om het mogelijk te maken om het systeem in offline modus ook te kunnen gebruiken. Er is voor de eerste versie gekozen om zo snel mogelijk de huidige TSM 1-op-1 te converteren naar het nieuwe systeem. De implementatie van het framework was nog niet begonnen.

Uit het gesprek is gebleken dat de scope van het project van Hans Tunnissen kleiner is dan die van ons. We hebben hierom besloten om onze eigen weg op te gaan. Hans Tunnissen heeft wel beloofd een aantal documenten te mailen (De opgestelde Requirements en 2 voorbeelden van een HTML TSM document).

Conclusie:
Het Web-TSM systeem dat vanuit HQ opgezet gaat worden is voor ons niet praktisch. Het enige dat er voorlopig (in de eerste versie) gebeurd is dat de TSM pagina's omgezet worden naar HTML en dan ingeladen worden. Ons project gaat om de TSM (en bijbehorend de SDS en eventueel STAR-Tools) te innoveren / integreren. Hiervoor moet de TSM aangepast worden en dat valt niet binnen de scope van het Web-TSM project van HQ. .

Gesprek Marvin Brunner over project - 15 juli
Na het gesprek met Hans Tunnissen en Rene Titulaer is er nog uitgebreid overleg geweest met Marvin over wat nu precies de probleemstelling is. In eerste instantie was het nog te vaag, maar gaandeweg het gesprek werd het steeds meer duidelijk welke richting de beste (en gewenste) is. Daarbij werd direct besloten dat medewerking aan het Web-TSM systeem van Océ HQ geen nut heeft, aangezien dit niet de innovatie op zal leveren waar (o.a.) Marvin naar op zoek is.

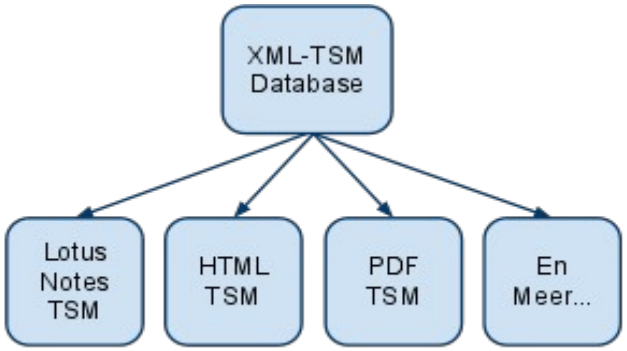
De TSM in huidige vorm *moet* (op enig moment) omgevormd worden naar een overzichtelijkere vorm, waarbij de manier waarop de Service Technicus het TSM raadpleegt de leidraad zal vormen. We hebben in dit gesprek 5 "entry-points" voor de TSM opgesteld.

De conclusie van het gesprek was dan ook dat er niet meegewerkt gaat worden aan een systeem dat er (momenteel) alleen op gericht is om het huidige TSM systeem (uit Lotus Notes) te converteren zonder aanpassingen te maken aan de structuur van de TSM zelf. Toen dit helemaal duidelijk was heeft Marvin nog wat tips gegeven over wat er bijvoorbeeld aangepast zou kunnen worden en waar naar gekeken kon worden.
Er is al eerder onderzoek gedaan naar de TSM en het bijbehorende verslag kan bestudeerd worden om te zien wat er voor bruikbare informatie uit te halen valt.

Hoofdstuk 2 - Het Technical Service Manual

Het Technical Service Manual (TSM) is een database waarin alle kennis staat die Océ heeft over haar eigen printers in een (al dan niet) makkelijk doorzoekbare vorm gerepresenteerd. De TSM wordt gebruikt door service technici om de kennis die ze zelf van printers hebben aan te vullen / op te zoeken. In dit hoofdstuk zullen wij alle kennis die wij opgedaan hebben over de structuur van het TSM bespreken. Belangrijk om te weten is dat het hier in alle gevallen gaat om onze interpretatie van informatie. Over het algemeen zal dit kloppen met de werkelijkheid, maar er is geen volledige zekerheid.

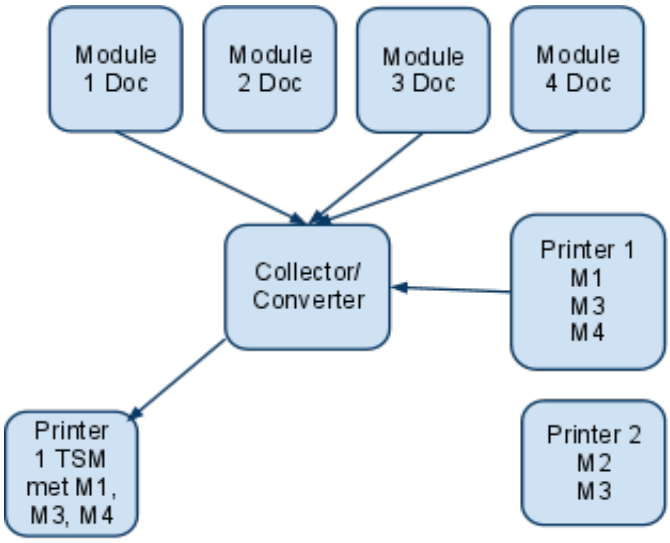
Aan de basis van het TSM staat de XML database (XML-TSM). In de XML-TSM staan de documenten die in het TSM horen samen met metadata weergegeven. Deze metadata geeft aan wat de (delen van) documenten voorstellen. Hierdoor kunnen XML-converters de XML-TSM exporteren naar elke vorm die de klant/servicemonteurs wensen te hebben. In feite is het dus een Content Management Systeem (CMS). In figuur 2.1 zie je als voorbeeld Lotus Notes, HTML en PDF weergegeven.



Figuur 2.1. Schematische weergave van hoe de verbindingen tussen de XML-TSM en andere TSM's zitten.

Modulaire opbouw

Océ is een bedrijf dat veel verschillende soorten printers verkoopt en onderhoudt. Voor elke printer moet er een specifieke TSM zijn, die alle kennis bevat die Océ heeft over dat printertype. Tussen verschillende printertypes worden er vaak modules hergebruikt. 2 zwart-wit printers die papier op verschillende manieren 'finishen'[5] zullen dezelfde printer module gebruiken en verschillende 'finish'-modules. Deze printers zullen voor de printer module dezelfde installatieprocedures kennen en dezelfde problemen zullen voorkomen. Kortom, het is nodig dat er in de XML-TSM ook veel documentatie herbruikt kan worden. Het Internationaal Trainingscentrum (ITC), verantwoordelijk voor het up-to-date houden van het TSM, heeft dit als volgt geïmplementeerd: Voor elke printer wordt er een printerconfiguratie file bijgehouden. In deze file wordt opgeslagen welke modules een bepaalde printer heeft. In Figuur 2.2 is te zien dat er een printertype Printer 1 is met de modules 1, 3 en 4, daarnaast is er een printertype Printer 2 met de modules M2 en M3.
Stel het is nodig dat het TSM van Printer 1 gegenereerd wordt in de vorm van een Lotus Notes database. Als eerste wordt de documentatie van verschillende modules



Figuur 2.2. Converter en Printer specifieke TSM

[5] Met 'finish' wordt bedoeld de processen die gebeuren nadat de printer module klaar is met printen, denk hierbij aan het vouwen/snijden van de printjes (Book-finish) en meer.

verzameld. Aangezien wij hier geen officiële naam voor kennen noemen we dit proces “Collection”; het uitvoerende programma noemen we de Collector. De Collector zal als eerste de printerconfiguratie file uitlezen, hierin vindt hij dat Module 1, 3 en 4 bij Printer 1 horen. Hij zal alle documentatie die bij deze 3 modules hoort bij elkaar zoeken en gebundeld presenteren aan de Converter. De Converter zal de XML bestanden uitlezen en in de vorm van een Lotus Notes database gieten.
De precieze details van hoe de Converter XML omzet in een Lotus Notes Database file is ons niet duidelijk geworden. Als het nodig blijkt te zijn voor verdere voortzetting van het project zullen wij hier zeker naar informeren.

Distributie

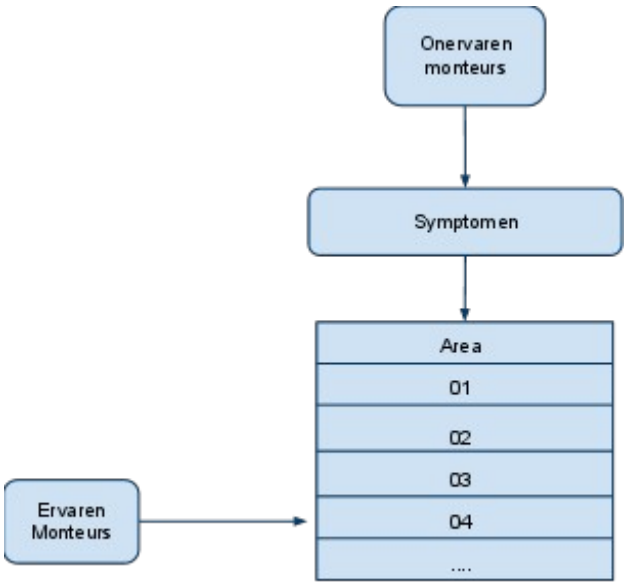
Distributie van het TSM gebeurt, in het geval van Lotus Notes TSM, door middel van nachtelijke ‘Pushes’. Service Laptops met het TSM worden elke avond aan een poort gehangen waarna de nieuwste versie van het TSM automatisch geupload wordt naar de laptops. Dit duurt ongeveer een uur, aangezien van elk type printer het TSM overgezet moet worden, wat in totaal om een paar gigabyte aan data gaat. Andere TSM’s (o.a. PDF) hebben geen reguliere updates en worden op verzoek van klanten verspreidt.

Annotaties

Een van de belangrijkste features van Lotus Notes is het gebruik van annotaties. Annotaties zijn aanpassingen die de gebruiker van het TSM aan documenten kan maken om een fout te corrigeren of zelf een aantekening te maken. De gebruiker kan ervoor kiezen om zijn annotatie door te spelen aan zijn lokale Service Product Manager (SPM), indien hij de annotatie belangrijk genoeg acht. De SPM zal hierop reviewen of deze annotatie belangrijk genoeg is om naar andere technici binnen zijn Operating Company (OpCo) door te spelen, of zelfs gelijk door te sturen naar HQ (indien erg belangrijk). Een SPM is over het algemeen iemand die heel veel kennis heeft van printers van Océ en als zodanig in een positie is om deze annotaties op waarde te beoordelen. Het ITC zal, in opdracht van HQ, deze annotaties verwerken in het systeem en deze aangepaste documenten aanmerken als ‘Review’. Hierna zal een SPM controleren of de informatie in het document juist is. Als dit het geval is, dan zal de SPM het document aanmerken als ‘Ready to Release’, waarop het document in de volgende push meegenomen zal worden. Documenten die gepusht zijn worden aangemerkt als ‘Released’.

TSM Documenten Structuur

Het ITC houdt zich bezig met het bruikbaar maken en houden van het TSM, voor zowel ervaren als onervaren monteurs. Ze doen dit door 2 entreepunten in de documentatie te bouwen, zoals te zien is in figuur 2.3 voor diagnose van printers. Dit systeem wordt binnen het hele TSM toegepast. De “vertical entry” wordt gebruikt door onervaren monteurs. Zij komen na een gesprek met de klant achter een aantal symptomen van het probleem. Door middel van de Symptom classification in het TSM zoeken zij stapsgewijs naar een symptoombeschrijving die past bij de symptomen zoals de klant ze aan de monteur heeft verteld. Uiteindelijk zullen ze, na het volgen van de pagina’s in het TSM, uitkomen bij het juiste onderdeel (Area) van de printer, bijvoorbeeld area 04.
Een ervaren monteur heeft een gesprek met de klant, en kan op basis van zijn ervaring direct de juiste diagnose stellen. In dit geval zal hij direct door hebben dat het probleem wel in area 04 moet zitten en zal via zogenaamde “horizontal entry” in 1 stap naar de juiste pagina gaan, waarna de informatie die daar staat hem helpt met het repareren / lokaliseren van het onderdeel. Hierbij wordt wel opgemerkt dat zeker de meer ervaren monteurs minder gebruik maken van de TSM en vanuit hun ervaring direct (kunnen) gaan sleutelen om het probleem op te lossen.



Figuur 2.3. Schematische weergave van hoe ervaren monteurs TSM gebruiken tegenover onervaren

Document Templates

Alle pagina’s binnen de TSM krijgen vanuit de XML-database een vaste structuur (template), zodat ze qua structuur consistent zijn. Er zijn verschillende soorten pagina’s en elk type pagina heeft een eigen template, bijvoorbeeld een template voor o.a. installatiepagina’s en diagnose pagina’s. Deze templates zijn gevormd uit een aantal velden waarin bepaalde blokken informatie geplaatst kunnen worden. Voor deze templates wordt veelvuldig gebruik gemaakt van het hergebruiken van blokken tekst.
De volgorde van de velden ligt vast en kan niet gewijzigd worden. Er kan wel voor gekozen worden om bepaalde blokken weg te laten als die voor een specifieke pagina niet nodig zijn, maar elk type blok, kan maar op exact 1 plaats neergezet worden binnen de pagina. Bijvoorbeeld het description block, bij een diagnose pagina, zal altijd bovenaan staan en dus zal de description zelf altijd bovenaan staan.

Hoe elk template er *precies* uit ziet is nog niet duidelijk, ook is het nog niet duidelijk welke structuur er binnen verschillende blocks bestaat. Het is wel van belang om hier achter te komen. We moeten namelijk weten hoe deze structuur eruit ziet voordat we weten wat we binnen het nieuwe TSM met deze documenten kunnen doen.

Plannen tot verbeteringen TSM

Na het analyseren van hoe het TSM in elkaar zit zijn wij tot de conclusie gekomen dat deze eigenlijk niet specifiek gemaakt is voor de service monteur. Er wordt wel een klein beetje rekening gehouden met het ervaringsniveau van monteurs, door de horizontale en verticale entry-points, maar er wordt toch heel veel aan eigenlijk statische pagina’s vastgehouden. Waarschijnlijk komt dat doordat er zo lang in Lotus Notes is ontwikkeld, dat Lotus Notes de structuur van het TSM ging dicteren in plaats van het faciliteren van het TSM.

Vanuit HQ zijn er plannen om het TSM aan te pakken, deze effort is vooral gericht op het zo snel mogelijk af te stappen van het TSM in Lotus Notes. Océ gaat binnenkort fuseren met Canon en hierbij willen ze

o.a. niet dat Canon geforceerd wordt om een licentie op Lotus Notes te nemen, alleen maar om het TSM van Océ printers te kunnen bekijken.
De focus in dit project is dan ook de huidige XML-TSM structuur te behouden en een web applicatie te ontwikkelingen die het TSM 1:1 overzet van Lotus Notes naar een HTML vorm. Er is veel druk vanuit HQ om deze versie 1.0 zo snel mogelijk te ontwikkelen. Als deze versie af is zal er in latere versies extra functionaliteit geïmplementeerd worden.

Wij vinden dit een minder praktische manier van werken. Om de fundamentele 'flaws' in het TSM aan te pakken moet je vanaf het begin af aan met een nieuw ontwerp beginnen. Je kunt niet een eerste versie van het TSM zonder innovatie in het oog maken en daarna verwachten dat er nog heel veel aan te passen valt. Wij hebben daarom gekozen om een nieuwe weg in te slaan wat betreft het ontwerp van het TSM. Deze zullen wij voor zover we het uitgewerkt hebben toelichten.

Nadat we gekeken hebben naar hoe een service monteur het TSM gebruikt, zijn we uitgekomen op 5 'entry-points', namelijk: diagnose via symptomen, diagnose via error codes, (preventive) maintenance, installation en modification. Dit zijn fundamenteel andere entry-points dan ze nu in het TSM verwekt zitten. Die andere entry-points gingen over het verschil tussen ervaren en onervaren monteurs. Onze entry-points gaan over met welke actie in gedachte de monteur het TSM gebruikt. Er is een kleine overlap tussen onze entry-points en die van het huidige TSM, bijvoorbeeld de diagnose via symptomen zal vaker gebruikt worden door onervaren dan ervaren monteurs, omdat een ervaren monteur vaak direct een idee heeft wat de oorzaak van het probleem is.

Elke entry-point heeft specifiek een andere manier van representeren van het TSM nodig. Voorlopig hebben wij deze manieren geïdentificeerd:

Diagnose via error codes:
Een printer geeft een error code en hierbij hoort een specifieke pagina. Een goede manier om de TSM via deze manier binnen te komen is door het een link te geven vanuit het SDS (waar de error log in staat) naar de specifieke pagina van het TSM. Hierbij kan in een soort gedetailleerd error scherm informatie gegeven worden over hoe vaak de error al is voorgekomen, wanneer deze is voorgekomen, maar ook welke onderdelen vaak de schuldige zijn en een directe link naar de pagina die bij dit onderdeel hoort. De TSM wordt hierbij op een totaal nieuwe manier gebruikt. De XML-TSM levert dan specifieke relevante informatie (as needed), in plaats van alle informatie in zijn database te leveren, omdat het er is.

Diagnose via symptomen:
Een symptoom kan bijvoorbeeld zijn dat er strepen zichtbaar zijn op de geprinte pagina. Hiervoor is het praktisch om afbeeldingen weer te geven i.p.v. via tekst te benoemen. In de huidige TSM wordt er gesproken over bijvoorbeeld: "Strepen in de x-richting" en "Strepen in de z-richting". Het is voor een Service Monteur veel makkelijker om direct te vergelijken met afbeeldingen, dan om eerst een aantal menu's door te lopen. De huidige TSM bevat sublagen voordat er überhaupt iets te vergelijken is, waarbij sublagen soms slechts 1 volgende entry hebben, wat de boom diep maakt, maar wat geen functie heeft.

Preventive Maintenance:
Er zijn bepaalde onderdelen in een printer die een bepaalde levensduur hebben, zogenaamde expendables. Dit is een standaard taak in het repertoire van een service monteur, voor elk onderdeel wordt een counter bijgehouden. Als deze een bepaalde threshold waarde overschrijdt ziet de monteur dat in een lijstje en kan hij ervoor kiezen om het onderdeel te vervangen. Net zoals bij error codes zou je een gedetailleerd scherm over deze taak weer kunnen geven met o.a. een link naar de TSM pagina die bij die part hoort en informatie over hoe vaak het al vervangen is, etc. Een dergelijke parts pagina ziet er over het algemeen hetzelfde uit als een installatie pagina. De structuur van de installatie pagina's, zoals wij die voor ogen hebben op het moment, leggen we bij "Installation" uit.

Installation:
Voor het installeren van een printer, of onderdelen ervan, is momenteel binnen de TSM een hoeveelheid pagina's beschikbaar, maar het is lastig door deze pagina's te navigeren. Lotus Notes opent namelijk voor elke link waar je op klikt een nieuwe tab. Met meerdere links per pagina (en meerdere links per pagina die je opent, enz.) raak je snel de weg kwijt.
Het is dus handig als a) alle pagina's in dezelfde tab blijven, b) er per stap die je uitvoert bijgehouden wordt of je hem al gedaan hebt of niet. Indien nodig kan je dan een deel-stap aanklikken en daarvan ook per stap bijhouden of je hem al gedaan hebt etc.

Modification:
Een modificatie is een aanpassing die een bepaald aspect van de standaardconfiguratie van een printer aanpast. Dit kan onderverdeeld worden in verplichte modificaties, wat vooral modificaties zijn die met veiligheid te maken hebben. De andere categorie bevat modificaties die bijvoorbeeld het gebruikersgemak of de print kwaliteit verhogen.
Om een overzicht te geven van welke modificaties geïnstalleerd zijn en welke niet, kan je een aparte tab maken waar al deze modificaties in een lijst samengevat worden met hun status erbij. De status zou dan zijn of ze geïnstalleerd zijn of niet, en hun 'prioriteit' (verplichte modificaties hebben een hogere prioriteit). Deze lijst bevat links naar de bijbehorende pagina in de TSM, met het installatie gedeelte hetzelfde als bij de installation entry-point.

Literatuurlijst

De literatuurlijst is voorlopig nog leeg. Wij hebben al onze informatie uit gesprekken gehaald met collega's bij Océ en eigen onderzoek (vooral naar de software waarin de TSM draait)

1) Brave NUI World - Designing Natural User Interfaces for Touch and Gesture - D. Widgor, D. Wixon.

Appendix B - Plan van Aanpak

Stage bij Océ

Gemaakt door:
Erwin Haasnoot
Jos Kuijpers

Begeleider Océ:
Marvin Brunner

Begeleider TU Delft:
Peter van Nieuwenhuizen

Inhoudsopgave

Voorwoord

Dit Plan van Aanpak beschrijft de plannen en eisen voor het Bachelorproject *Touch screen user interface design for Service*. Dit project is uiteindelijk veranderd in een algemeen *improve service software* project. Het project vindt plaats in het Software Engineering 1 departement van de Research & Development afdeling van Océ.

1. Introductie

1.1 Aanleiding

Voor het oplossen van problemen die optreden bij de printers, wordt gebruik gemaakt van een combinatie van een Technical Service Manual (TSM) en een Service Diagnostic System (SDS). Het SDS systeem is in overgang van een active-x applicatie naar een web-based systeem. De TSM echter hangt nog steeds aan Lotus Notes vast en dat beperkt de portabiliteit van het systeem aanzienlijk. Binnen ons project zullen wij uitzoeken *hoe* het TSM verbeterd kan worden en *wat* wij hier daadwerkelijk aan kunnen doen.

1.2 Accordering en bijstelling

De goedkeuring van het Plan van Aanpak geschiedt via onze begeleider, Peter van Nieuwenhuizen. Elke twee weken zal er een voortgangsrapport gestuurd worden naar de begeleider. Het PvA en deze rapporten vormen uiteindelijk het volledige PvA.

1.3 Toelichting op de opbouw van het plan

De verdere opbouw van dit document zal er als volgt uit zien:

- Hoofdstuk 2 bespreekt alle details van de opdracht
- Hoofdstuk 3 bespreekt hoe wij dit project zullen aanpakken, wat wij in welke fase doen
- Hoofdstuk 4 bespreekt hoe de Projectinrichting zit, wie is verantwoordelijk voor welk deel van de opdracht en wie moeten we daarvoor benaderen.
- Hoofdstuk 5 bespreek hoe de planning precies zal zitten
- Hoofdstuk 6 bespreekt hoe de opdrachtgever de kwaliteit van het project zal kunnen waarborgen.
- Hoofdstuk 7 bespreekt de risico's die inherent zijn aan het project, en hoe we daarmee om zullen gaan. Oftewel, er gaat een keer iets niet zoals wij willen, wat dan?

2. Projectopdracht

In dit hoofdstuk wordt de opdracht afgebakend, door middel van het beantwoorden van de 'waarom', 'waarover' en de 'wat' vragen.

2.1 Projectomgeving

De Projectomgeving wordt gezien als de Service Monteur die op een Service Call reageert. De laptop die de monteur bij zich heeft bevat o.a. de TSM en de (software)Tools die nodig zijn om de volledige diagnose te stellen en het probleem op te kunnen lossen. Het probleem dat optreedt, is dat de TSM niet altijd overzichtelijk genoeg is voor de Service Monteur. Vooral ervaren monteurs hebben eerder de neiging te gaan sleutelen, daarbij gebruikmakend van hun jarenlange ervaring, dan dat ze het TSM gebruiken.[6] In principe is daar niets mis mee, maar het TSM zou goed kunnen helpen met het versnellen van dit proces, wat het op het moment niet doet.
Daarnaast worden per oplossing veelal meerdere onderdelen aangegeven waar de fout aan kan liggen. De volgorde is gebaseerd op de tijd die het kost om de diagnose te doen. Echter is er statistische data bekend waaruit blijkt dat een groot percentage van alle errors (+- 80%) veroorzaakt wordt door een zeer klein percentage van alle onderdelen (+- 5%).[7] De TSM bevat echter tests voor (praktisch) alle onderdelen die in theorie een error kunnen geven, dit is voor de overzichtelijkheid en efficiëntie niet altijd even praktisch.

2.2 Doelstelling project

De doelstelling van het project is om de TSM dusdanig aan te bieden aan de Service Monteurs zodat deze op een snelle en efficiënte manier de problemen op kunnen lossen. Oftewel: gebruiksvriendelijker maken van de TSM. Daarnaast zijn er nog mogelijkheden om de efficiëntie van distributie van het TSM te verbeteren.

2.3 Opdrachtformulering

Het ontwikkelen van een geïntegreerd TSM systeem dat de Service Monteurs ondersteunt bij het oplossen van problemen bij de printers.

2.4 Probleemstelling

Het eigenlijke probleem hebben wij gedefinieerd als: "Het TSM helpt niet bij het verkorten van service bezoeken". Hierop volgen een aantal problemen die er mede voor zorgen dat het TSM niet echt helpt. Deze vallen onder te verdelen in 2 categorieën: Problemen die voortkomen uit het gebruik van Lotus Notes en problemen die voorkomen uit de structuur van het TSM.
Lotus Notes:

- Geen portabiliteit tussen systemen
- Elke TSM voor elke printer moet als geheel op elke laptop staan
- Weergave van volgordes, bijvoorbeeld bij de lijst met onderdelen die te maken kunnen hebben met een bepaald probleem of een lijst die je krijgt als je zoekt, is gedeeltelijk Random (zit geen duidelijke logica achter). Is volgens Pieter Dielissen gebaseerd op de tijd die nodig is om een onderdeel te testen.
- Er is geen mogelijkheid tot inzoomen op onderdeel-plaatjes. Lastig om onderdelen exact te lokaliseren als er heel veel onderdelen in een klein gebied zitten. Heeft ook te maken met of een onderdeel voor of achter een bepaalde plaat zit. Technicus bestelt alles in de omgeving...
- Geen manier om bij te houden op welke stap je precies zit, als je een stappenplan aan het volgen bent.

[6] Internal Memo & Interview met Marvin Brunner.

[7] Brunner, M. 2009. Analysis Figures from the Field - Internal Memo.

- Tab-structuur niet altijd wenselijk, elke aparte link opent een nieuwe tab. Als er vanaf 1 bepaalde pagina een diepte van 5 pagina's is, raak je heel snel de weg kwijt. Mede mogelijk gemaakt door de absentie van een manier om de gevolgde stappen bij te houden.
- Knipperende links zijn enorm irritant en leiden tot vroegtijdig klikken
- Zoekopdracht werkt niet goed. De zoekopdracht moet exact kloppen, als er aan het begin en/of eind karakters/cijfers missen, wordt er niets gevonden.
- Licentie lotus notes is duur

TSM-structuur

- Er zijn templates, maar deze worden niet altijd aangehouden bij het opzetten van pagina's
- Enkele duizenden megabytes aan data moeten verstuurd worden naar elke service laptop elke keer dat er een update is. (Pieter)

2.5 Op te leveren producten en diensten

Op te leveren product is een prototype van het geïntegreerde TSM systeem. Daarnaast zullen wij de volgende documenten opleveren:

- Design Document
- Plan van Aanpak
- Oriëntatie Verslag
- Eindverslag

2.6 Eisen en beperkingen

De volgende eisen zijn van toepassing:

- Een aantal huidige wenselijke functionaliteit blijft behouden, dit houdt in:
    - Het annotatie systeem blijft behouden
    - Ervaren monteurs moeten snel en efficiënt naar het document dat ze nodig hebben kunnen komen (horizontal entry). Dit systeem wordt echter wel op een andere manier opgezet dan in de huidige vorm. De huidige vorm bevat 2 hoofd-classificaties (Area en Symptom) en daar hangt alles onder. In het ontwerp van het prototype wordt een onderscheid gemaakt in 5 classificaties (Errors, Symptoms, Modifications, Installations en Preventive Maintenance).
    - Onervaren monteurs moeten, indien ze het nodig hebben, aan de hand genomen worden en door het TSM geleidt kunnen worden. (vertical entry)
- Vanuit de printer direct naar een Error Pagina geleid worden, zonder dat er gezocht hoeft te worden.
- Het systeem moet onderhoudbaar zijn voor anderen, dit betekent dat code overzichtelijk moet zijn, goed gecomment en gedocumenteerd en ten slotte dat methoden binnen de code een beperkte lengte hebben (Helpt voor het begrijpen en maakt het overzichtelijk).
- Uiteraard zal het prototype niet alle functionaliteiten bevatten die tijdens het design zijn opgesteld. Daarvoor is de periode te kort. Dit betekent dat het mogelijk moet zijn om deze functionaliteiten op een relatief eenvoudige wijze toe te kunnen voegen aan het prototype. Dit gaat gepaard met de onderhoudbaarheid. Als het goed onderhoudbaar is zal het ook goed uitbreidbaar moeten zijn.

2.7 Cruciale succesfactoren

Cruciale factoren om het project tot een succes te brengen is het begrijpen van hoe het huidige TSM-systeem in elkaar zit en hoe de wensen van de verschillende partijen (de Service Monteurs, het Training Centrum en HQ) verwerkt kunnen worden tot een degelijk systeem.

De kwaliteit van ons prototype hangt voor een groot deel af van de hoeveelheid pagina's die goed

ingelezen en gedisplayed kunnen worden. Hiervoor zullen wij XML Schema's opzetten die de XML pagina's beschrijven zoals wij ze hebben. Ook zullen we een test-suite maken die voor elke pagina checkt of de pagina goed ingelezen kan worden, en zo nee, waar het fout gaat.

3. Aanpak

Onze aanpak is in 4 fases onderverdeeld; de oriëntatiefase, de designfase, de implementatie / prototypefase en de testfase. Tussen de implementatiefase en de testfase is geen harde scheiding, maar naarmate het project vordert zal er meer focus gelegd worden op het testen en minder op het implementeren.

3.1 Oriëntatiefase

Deze fase is bedoeld voor het vergaren van kennis, wij hebben dit als volgt gedaan: Ten eerste zullen wij veel informatie verzamelen over hoe het 'servicen' van printers in elkaar steekt, vooral over de software kant van service. Uiteindelijk hebben we ons toegespitst op het TSM en de pijnpunten van dit systeem proberen bloot te leggen. Om daarna uit te zoeken wat de huidige ontwikkelingen zijn wat betreft het verbeteren van het TSM.

3.2 Designfase

In deze fase zullen wij een design doc schrijven waar verschillende manieren van design representatie in uitgewerkt worden naar gelang de situatie daarom vraagt. De GUI zullen we bespreken met Fred de Jong (een interaction designer) om een idee te krijgen wat goede GUI design is en wat niet. Deze fase zal ervoor zorgen dat wij alles in de implementatie fase makkelijk kunnen implementeren en vooral ook goed taken kunnen verdelen. In deze fase worden ook sequensendiagrammen gemaakt om het programmeerproces makkelijker te kunnen doorlopen.

3.3 Implementatiefase

In deze fase vindt de implementatie van het verbeterde TSM plaats. Dit wordt een nieuw systeem. Het product dat uit deze fase voortvloeit, zal een prototype zijn dat de kracht en mogelijkheden van het nieuwe TSM zal laten zien. Daarnaast zal deze modulair opgebouwd en makkelijk uit te breiden en te onderhouden zijn. We designen het volledige systeem, al zal later pas blijken hoeveel er daadwerkelijk geïmplementeerd kan worden. De rest zal dan door een andere partij naderhand geleverd kunnen worden als er besloten wordt om door te gaan met ons ontwikkelde systeem.

3.4 Testfase

In deze fase wordt het product uitvoerig getest in de werkelijke omgeving en worden eventuele bugs en complicaties verholpen.

4. Projectinrichting en voorwaarden

4.1 Projectinrichting

- De *organisatie* rond dit project is als volgt ingericht:
    - Drs. P.R. van Nieuwenhuizen, Mediamatica, TU Delft, Projectbegeleider
    - Marvin Brunner, Software Engineering I, R&D Océ, Projectbegeleider, Opdrachtgever

- Het *personeel* dat betrokken is, is zijn de opdrachtgever, de afdeling R&D, de afdeling ITC en de afdeling Service (waar de Service Monteurs onder vallen). Deze afdelingen zijn betrokken bij de ontwikkelingen en gebruik van de TSM.
- Onder de *Administratieve* procedures vallen het Plan van Aanpak, het Oriëntatieverslag, de tweewekelijkse rapporten, een eindverslag en eindpresentatie.
- *Financiering* is niet van toepassing bij deze opdracht, aangezien alles al tot onze beschikking staat.
- Voor *Informatie* zal er regelmatig contact zijn met de projectbegeleider en de instanties die bij dit project van toepassing zijn. Eventuele vragen kunnen zo snel beantwoord worden.
- De benodigde *techniek* bestaat uit de ter beschikking stellen van o.a. Lotus Notes en de TSM. Verder zijn er werkplekken ingericht waarop alle benodigde software aanwezig is (of aangevraagd en geleverd wordt).

4.2 Voorwaarden aan opdrachtnemer (wijzelf)

De volgende voorwaarden worden gesteld aan de opdrachtnemer:

- Het project wordt binnen ongeveer 10 ½ week afgerond met een eindproduct.
- De geschreven software is goed gedocumenteerd zodat verdere uitbreiding mogelijk is.
- Het project wordt afgesloten met een eindverslag en een presentatie op de TU in Delft.

4.3 Voorwaarden aan opdrachtgever

De voorwaarden die gesteld zijn aan de opdrachtgever:

- De mogelijkheid bieden om het TSM systeem te onderzoeken en (eventueel) vervolgonderzoek te analyseren.
- Informatie verstrekken wanneer dit nodig geacht wordt.
- Wanneer nodig programmacode en software beschikbaar stellen.

5. Planning

Week 1-2 (4 juli - 15 juli)

Oriëntatie:
In deze fase wordt er georiënteerd op de opdracht en is het belangrijk om zoveel mogelijk informatie te vergaren om een idee te krijgen wat er nu precies moet gebeuren voor de opdracht. Hiervoor moet het huidige TSM-systeem onder de loep genomen worden en contact zoeken met de verschillende onderdelen van Océ die te maken hebben met het TSM (HQ, Internationaal Training Centrum (ITC), Service)

- Idee krijgen van service software van Océ
- Opdracht specificeren
- Plan van aanpak schrijven
- Orientatieverslag schrijven

Week 3-4 (18 juli - 29 juli)

Ontwerp:
In deze fase wordt het ontwerp voor het systeem opgezet. Hiervoor zijn requirements nodig, schetsen voor de GUI en klassendiagrammen voor de benodigde klassen.

- Requirements opstellen voor het ontwerp
- Klassendiagram / Sequence Diagram maken met bijbehorende beschrijvingen methodes
- GUI ontwerpen met behulp van storyboards
- Bepalen ontwikkelingsmethode
- Bespreken ontwerp met begeleider en opdrachtgever

Week 5-10 (1 augustus - 2 september)

Implementatie en Testen:
In deze fase wordt het daadwerkelijke product geïmplementeerd. Voor ons systeem moet een GUI gemaakt worden met de bijbehorende schermen om de informatie te visualiseren. Verder is er een manier nodig om de XML bestanden (van het Web-TSM) in te lezen en te verwerken. Aan het eind van deze fase wordt ook de code ingeleverd bij de Software Improvement Group (SIG) en wordt een begin gemaakt met het eindverslag.

- Implementeren GUI
- Implementeren XML-Reader(s)
- Implementeren Programma binnen GUI
- Implementeren test cases (Junit, en veel, veel meer!)
- Aanvang Eindverslag
- 31 Augustus: vrijgave eindverslag regelen met afdelingshoofd (John Kesseler)
- 24 Augustus: Code inleveren SIG (1e keer)

Week 11-12 (5 september - 16 september)

Afronding:
De laatste fase van het project bevat de afhandeling van het commentaar van SIG, het afronden van het eindverslag (en de vrijgave hiervan regelen) en de code voor de 2e maal op te sturen naar SIG.

- Afronden Eindverslag
- Afhandelen feedback SIG
- 6 september: Vrijgave eindverslag regelen met afdeling Corporate Patents.
- Demo maken voor presentatie.

- Inleveren eindverslag (als dat nog niet gedaan is)
- Inleveren code SIG (2e keer)

Week x (x september - y september)

Op een tijdstip ergens na de Afrondingsfase vinden er nog 2 presentaties plaats: 1 op Océ en 1 op de TU Delft.

- Eindpresentatie bij Océ
- Eindpresentatie op de TU Delft.

6. Kwaliteitsborging

6.1 Productkwaliteit

6.1.1 Functionaliteit

Uiteraard moet het programma correct functioneren. Het programma zal verder alle functionaliteit die gewenst is binnen Lotus Notes moeten hebben. Hierbij denken wij aan o.a. annotaties en structuur van documenten behouden. Ook zullen wij moderne technieken toepassen om het TSM te verbeteren, hierbij kan je denken aan het gebruik van SVN bij het distribueren van het TSM.

6.1.2 Betrouwbaarheid

De betrouwbaarheid is samen met de bruikbaarheid een belangrijk deel van het eisenpakket dat opgesteld is (wordt). Als de betrouwbaarheid zou falen, dan zal de monteur er niets aan hebben en dan zou het geen verbetering zijn op de bestaande software.

6.1.3 Onderhoudbaarheid

Aangezien de ontwikkeling van printers niet stil staat zal de ontwikkeling van de TSM ook niet stil blijven staan. Het is niet mogelijk om nu al te voorzien wat voor toevoegingen er in de toekomst komen, dus is het belangrijk dat de software goed gedocumenteerd is, zodat deze aangepast kan worden om de toevoegingen goed op te kunnen vangen. Daarnaast is het belangrijk om alles wat we maken modulair op te zetten, zodat bepaalde stukken zonder problemen vervangen kunnen worden.

6.1.4 Efficiëntie

De efficiëntie van het product is voornamelijk van belang voor de Service medewerkers in het veld, die, als ze de TSM gebruiken, zo snel mogelijk de pagina's kunnen vinden die nodig zijn om een bepaald probleem op te kunnen lossen.

6.1.5 Bruikbaarheid

Het is belangrijk dat de Service Monteurs het gevoel hebben dat het nuttig is om de TSM te gebruiken bij een Service Call. Dus is het van belang dat de gebruiksvriendelijkheid hoog ligt en dat het programma duidelijk en overzichtelijk is voor zowel ervaren monteurs als voor de minder ervaren monteurs. Er moet voor onervaren monteurs een leidraad (vertical approach) zijn, terwijl ervaren monteurs direct naar de specifieke informatie moeten kunnen die ze nodig hebben (horizontal approach).

6.1.6 Portabiliteit

De keuze om niet meer volledig afhankelijk te zijn van Lotus Notes is mede vanwege het verlangen om een betere portabiliteit te hebben en om het gebruik van de service laptop langzaam uit te faseren. Als de TSM op de printer zelf draait zal de laptop niet meer noodzakelijk zijn en kan daarmee veel tijd bespaard worden (aansluiten, opstarten, enz.).

6.2 Voorgestelde maatregelen

Een aantal acties zijn mogelijk om de kwaliteit te waarborgen:

- Gesprekken met de partijen die betrokken zijn bij de ontwikkeling van / het gebruik van / het onderhouden van de TSM.
- Degelijk gedocumenteerde code.
- In een later stadium de testomgeving te laten testen door enkele Service Monteurs die zelf regelmatig de huidige TSM gebruiken. Hier zou het moeten gaan om een goede mix van Service Monteurs (ervaren en onervaren). Hierdoor kunnen we meer feedback krijgen.

7. Risk Management

Om te kunnen gaan met risico's zullen wij deze eerst identificeren. Per fase zullen we de risico's, met een redelijke kans van gebeuren, benoemen en hoe we ermee om zullen gaan mocht het fout gaan.

7.1 Orientatiefase

Risico: Meeloopstage service monteurs kan niet doorgaan
Wij hebben hier rekening mee gehouden door de meeloopstage dubbel te plannen. Wij lopen elk met een andere service monteur mee, dus mocht het bij 1 monteur niet doorgaan dan is nog de ander die wel doorgaat. Daarnaast kan er een andere datum gepland worden waarop het beter uitkomt. De kans dat beide meeloopstages niet door kunnen gaan en dat er op korte termijn geen vervanging geregeld kan worden schatten wij in op nihil.

7.2 Designfase

Risico: Fred de Jong heeft de komende maanden geen tijd voor een interview
De input van Fred zal zeker van waarde zijn, maar wij hebben zelf ook kennis van degelijk GUI design. Daarnaast bouwen we het modulair op, zodat elk onderdeel (inclusief het GUI) makkelijk te vervangen is.

Risico: Design van al bestaand systeem sluit niet aan op wat wij voor ogen hebben, en het valt redelijkerwijs ook niet aan te passen
Wij zullen de designer moeten overtuigen, of zelf overtuigd worden van het systeem dat er al is. Als dit niet mogelijk blijkt te zijn zullen we 'from scratch' een systeem op moeten bouwen.

7.3 Implementatiefase

Risico: Wij mogen onze code niet naar SIG sturen, vanwege issues met CP
Als uiteindelijk blijkt dat we onze code niet naar SIG mogen sturen, omdat er bv geen overeenstemming komt over het NDA, dan zullen wij zelf opzoek moeten gaan naar 'code controleurs'. Een mogelijkheid hier is het laten review door collega's van SE1, door xx (een bedrijf als SIG waar Océ al ervaring mee heeft), zelf automatische code checkers draaien. Alles wat mogelijk is om een onafhankelijke kwaliteitskeuring te krijgen.

7.4 Testfase

Risico: (Insert groep die mogelijk wil testen) mogen niet onze vernieuwde TSM testen, of ze hebben hier geen tijd voor
Als we de groep van mensen die mogelijk willen testen groot genoeg maken, dan zullen we genoeg redundantie in het testen hebben dat als er 1 groep uitvalt, het geen probleem is dat die uitvalt.

Appendix C - Requirements Doc

Alle requirements zoals wij die hebben opgemerkt / hebben gekregen / bedacht hebben, schrijven wij in dit Requirements Doc.

Functional Requirements
De gebruiker van het systeem moet de volgende acties kunnen ondernemen met het systeem:

Algemeen

- Het TSM updaten naar de nieuwere versie
- Het uitvoeren van SDS-testen en hier feedback van krijgen. Waar mogelijk deze feedback automatisch in het systeem verwerken.
- Annotaties aanbrengen op een pagina, of aangeven waar een fout staat
- Het moet mogelijk zijn om in te zoomen op afbeeldingen.
- Het persoonlijk inloggen op het systeem, zodat annotaties hierop gehangen kunnen worden.
- Vanaf een errorcode direct naar de juiste error pagina doorverwezen worden.
- Bijhouden welke modificaties zijn uitgevoerd (al dan niet automatisch).
- Statistiek vergaren, uploaden naar een centrale database, zodat anderen het weer kunnen downloaden. (Om snel informatie te verzamelen over veel printers).

Detailed Info Page

- Statistiek (trendanalyse) over errors kunnen zien.
    - Hoe vaak is een bepaalde error opgetreden en op welke tijdstippen.
- Lijst met Suspected parts / settings geordend weergeven, gebaseerd op statistieken van de parts en mogelijk de uitkomsten van test / checks. (fail rate)
- Test/checks moeten weergegeven worden en de uitkomst van zo'n Test moet verwerkt worden in de volgorde van Suspected Parts & Settings.

Parts Page

- Usagecounter van vervangen onderdelen resetten.
- Statistiek (trendanalyse) over parts kunnen zien.
    - Hoe vaak en wanneer is de part vervangen
    - Bij welke levensduur (maintenance counter stand) de part vervangen is.
- Optional: Het bijhouden van vervangen onderdelen

- Opzoeken waar een onderdeel zich bevindt in de printer.
- Opzoeken hoe een bepaald onderdeel vervangen moet worden.
- Opzoeken wat het nummer is van te bestellen onderdelen.
- Het opvragen van een elektrisch diagram dat bij een onderdeel hoort.
- Het opvragen van een "bouwtekening" van een onderdeel.
- Verschillende tests kunnen aanroepen om onderdelen uit te sluiten.
- Opvragen welke onderdelen de monteur in de auto heeft.

Test & Checks block

- Elke Test en elke check moet een duidelijke fail/success case hebben. In ieder geval duidelijk genoeg voor een servicemonteur, als het kan duidelijk voor een computer.
- Aangeven welke test/check gefaald is en welke niet.
- Parts & Settings linken aan een test, als een test faalt, verander de kans dat een part ermee gemoeid is.
- Geef aan per stap in de test/check of deze automatisch uitgevoerd kan worden of niet. (optioneel)

Modification Page

- Aangeven of specifieke modification al is uitgevoerd.
- Klasse van de modificatie weergeven.
- Serienummer van printer weergeven.
- Geef aan of een modificatie van toepassing is op deze printer.
- Het opvragen van de lijst met Modificaties.

Guided Diagnostics Page

- Image Quality issues makkelijk doorzoekbaar door plaatjes, ipv tekst weer te geven.
- Categorisatie niet meer op systemen waar de oorzaak vandaan komt. (maar bijvoorbeeld op gelijkheid van symptomen)

Installation

- Voor zover de gebruiker weet stappenplannen lineair kunnen volgen. Gebaseerd op een lineaire presentatie van de stappen.

Non-Functional Requirements

Deze requirements worden gemeten door het nieuwe systeem te vergelijken met het oude systeem en te bekijken wat het verschil in te nemen stappen is.

User interface and human factors

- De user interface moet intuïtief en gebruiksvriendelijk zijn.
- De gebruiker moet in zo min mogelijk handelingen zijn doel kunnen bereiken.
- Er moet rekening gehouden worden met de ervaring van de monteur, dit betekend dat de onervaren monteurs bij de hand genomen moeten kunnen worden, maar dat de ervaren monteurs op elk punt kunnen inspringen en uitspringen.
- De verschillende pagina's (o.a. Detailed Info Page, Modification page, parts page) moeten los van elkaar aangeroepen kunnen worden, een soort van state-loosheid.

Distribution TSM

- TSM distributie moet mogelijk zijn zonder dat de printer toegang heeft tot internet
- Data transport per update moet minimaal zijn

Hardware considerations

- De software moet portable zijn, moet kunnen draaien op elke gangbare hardware.

Performance characteristics

- Het systeem moet snel en soepel werken, zo min mogelijk laadtijden.

Error handling and extreme conditions

- Als iets van ons crasht, moet de SDS nog beschikbaar zijn.
- Als een pagina crasht, moet het makkelijk zijn om hier weer terug te komen.

System modifications

- Het moet goed onderhoudbaar zijn
- Het moet makkelijk zijn om extra functionaliteit toe te voegen waar wij in ons ontwerp geen rekening mee gehouden hebben.

Physical environment

- Het systeem wordt gebruikt in een omgeving die niet te controleren valt, dus alle data die nodig is moet in de printer zelf zitten of meegenomen kunnen worden door de monteur.

Constraint

- Er mag geen Lotus Notes gebruikt worden voor de implementatie
- Aangezien het TSM geïntegreerd gaat worden met het al opgezette SDS, zal er in ongeveer dezelfde ontwikkelomgeving gewerkt moeten worden.
- Er wordt hoofdzakelijk met Touch-Screens gewerkt, daar moet rekening mee gehouden worden bij het ontwerpen. (size van buttons, geen “scroll-overs”, e.d.)

Appendix D - Design Document

Stage bij Océ

Gemaakt door:
Erwin Haasnoot
Jos Kuijpers

Begeleider Océ:
Marvin Brunner

Begeleider TU Delft:
Peter van Nieuwenhuizen

Inhoudsopgave

Introduction

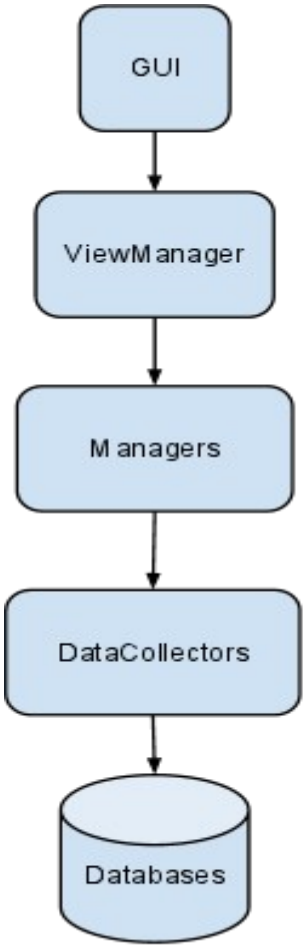
In dit document zetten we uiteen hoe we ons systeem hebben ontworpen. De onderwerpen die aan bod komen zijn:

- Lagensysteem
- Packages
- XML
- GUI-ontwerp
- Sequence Diagrammen

Hoofdstuk 1. Laagstructuur

Het systeem is onderverdeeld in een aantal lagen, te weten de volgende:

- de GUI, laat op het beeld de juiste pagina's zien en verzorgt de IO van de User.
- ViewManager, deze Manager verzorgt de communicatie tussen de GUI en de Data Collectors die de pagina's aanmaken.
- Managers, de managers maken de pagina aan die aangevraagd is door de ViewManager. Vraagt de informatie op bij de Data Collectors.
- Data Collectors, deze instanties halen de ruwe data op uit de bijbehorende bestanden (XML) en geven deze terug aan de Manager.

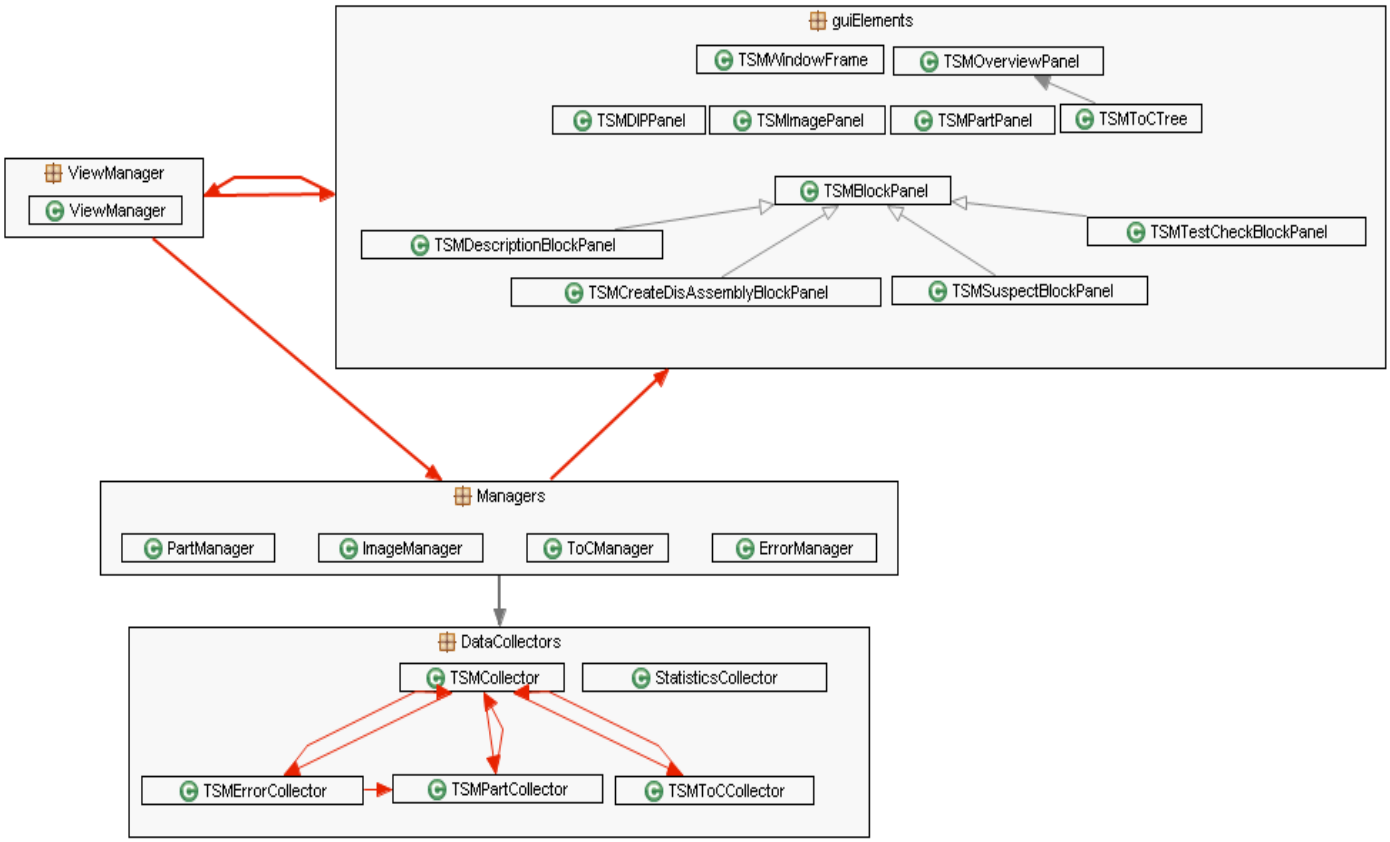


Figuur 1 globale opzet

1.1 Waarom hebben we voor deze opzet gekozen?

Het eerste waar we achter kwamen was dat ons TSM opgebouwd zou worden uit veel verschillende informatie bronnen (databases). De DataCollectors verzorgen een interface tussen deze databases en de rest van het programma. Als de opzet van 1 van de databases verandert, hoeft alleen de onderste laag van de DataCollectors aangepast te worden.
De Manager laag daarboven creëert de verschillende "Pages" die wij gedefinieerd hebben. Omdat Pages vaak informatie nodig hebben uit meerdere bronnen en er veel overlap is tussen de stukjes informatie die door elke page gebruikt wordt, hebben wij ervoor gekozen om deze Manager laag een aparte laag te maken.
De ViewManager zorgt voor juiste weergave van de verschillende pagina's als GUI elementen. Deze roept ook de Managers aan om pagina's voor 'hem' te creëren.
De beschikbare informatie convergeert uiteindelijk, door transformaties binnen de DataCollectors en Managers lagen, naar de ViewManager toe.

Hoofdstuk 2 - Package Diagram



Figuur 2: Dependancy Diagram

Hierboven (Figuur 2) staat een dependancy diagram waarin de huidige klassen zijn gespecificeerd en de packages waar ze onder vallen. De GUI is weggelaten, omdat de functionaliteit van deze klasse beperkt is. De guiStarter klasse die hier in zit heeft als enige functionaliteit het aanmaken van de GUI, alle andere functionaliteiten worden aangeleverd door de ViewManager.

2.1 ViewManager

Deze package bevat de klasse die de aanmaak van pagina's regelt wanneer de aanroep komt vanuit de GUI. Deze klasse staat tussen de GUI en de data in en zorgt er voor dat de juiste Managers aangeroepen worden wanneer een pagina aangemaakt moet worden.

2.2 guiElements

De guiElements package bevat alle klasses voor onze GUI. Wij hebben dit opgesplitst in 3 lagen. In de eerste laag vindt je de WindowFrame en OverviewPanel, deze zijn gemaakt om elke JPanel in te kunnen laden. Een windowFrame zit 'los' van het UI, als een internal frame.
De OverviewPanel zit vastgekoppeld aan het main UI. Dit panel bevat aan de linkerkant de Table of Contents en aan de rechterkant de huidige pagina die bekeken wordt.
In de 2de laag zitten 'pagina' panels. Deze bevatten een lijst met 'blockpanels'. De Blocks zijn stukken GUI die altijd gelijk zijn qua lay-out voor de verschillende pagina's. Parts en Errors (en nog meer) hebben bijvoorbeeld allemaal een description. Het is niet zinvol om dan voor elk van deze pagina's aparte description blocks te maken, terwijl het ook in 1 klasse kan.
In de 3de laag zitten de 'BlockPanels'. Voor elk van deze blokken is de 'headerpane' hetzelfde (bevat een titel en een aantal knoppen). Via overerving ontvangen alle subklasses (specifieke blokken) dezelfde headerpane, en definiëren ze allemaal een eigen contentpane.

2.3 Managers

Deze package bevat een viertal Managers die ieder toegang hebben tot een eigen deel van de TSM. Met de informatie die opgevraagd wordt, wordt vervolgens de pagina gebouwd en teruggestuurd naar de ViewManager.

- De PartManager vraagt alle data over een part op.
- De ImageManager vraagt afbeeldingen en diagrammen op van een part.
- De ToCManager maakt en beheert de Table-of-Contents.
- De ErrorManager vraagt alle data op die nodig is om een Error Pagina aan te maken.

Elke Manager roept een aantal DataCollectors aan om de informatie op te zoeken die nodig is om de pagina te kunnen maken.

2.4 DataCollectors

Deze package bevat een aantal klassen die ruwe data (in XML-formaat) uitlezen en deze omvormen tot een DOM-tree, dit is in feite een XML tussenvorm die wij gecreëerd hebben. De package geeft deze daarna terug aan de Manager die de Collector had aangeroepen.

Bij deze package hoort ook het XML-parser gedeelte dat nodig is om de input XML bestanden uit te lezen en vervolgens weg te schrijven naar een simpeler formaat. Het schema van hoe deze XML tussenvorm eruit ziet zullen we in hoofdstuk 3 beschrijven.

Hoofdstuk 3 - XML

Binnen het project is het van belang dat de TSM-database uitgelezen kan worden. Om te laten zien dat het project “aansluit bij de werkelijkheid” is er voor gekozen om de XML files van de HTML-TSM te gebruiken. Deze files geven dezelfde informatie weer als de TSM uit Lotus Notes.

Echter is de lay-out van deze XML-files dusdanig (zo worden bijvoorbeeld tabellen vak voor vak gedefinieerd) dat er gekozen is om een tussenvorm te gebruiken voor de XML. Hiervoor is een parser ontworpen die de XML bestanden van de HTML-TSM bestanden inleest en er precies uit haalt wat nodig is voor onze versie van de TSM. Alle informatie die verder niet nodig is wordt genegeerd.

Er zijn een aantal voordelen bij het gebruik van een XML-tussenvorm. Namelijk, het omzetten van het hybride lay-out/content ruwe XML (waar XML eigenlijk niet voor bedoeld is), naar een ‘puur’ XML, met alleen content. Daarnaast hoef je de zware XML transformatie (ruwe XML -> Swing GUI Elementen) niet meer uit te voeren zodra je een pagina al een keer hebt geopend. De XML-tussenvorm komt veel beter overeen met hoe Java Swing is opgezet, en hoe wij onze GUI als gevolg daarvan hebben opgezet.
Als derde punt zijn er in het huidige TSM geen part pages, dat wat wij part pages noemen zit verspreidt in het originele TSM over 3 verschillende pagina’s (en dus XML bestanden). Een deel zit in de TSM Error Pages, een deel in de ‘Parts List’ en mogelijk een deel in de dis-assembly Page (als deze bestaat voor die part). De manier waarop het verspreidt is en het feit dat er totaal geen link tussen de pagina’s zit (op het gebied van parts), maakt het enorm inefficiënt om uit het TSM een part page te halen. Daarom, zodra een error page door de parser uitgelezen wordt, maakt de parser ook gelijk part pages aan.

De design van onze XML-tussenvorm staat in appendix E, inclusief uitleg.

Transformatie
Momenteel is de enige transformatie die uitgevoerd wordt de Error Page transformatie.

In het begin zijn er 2 bestanden, de ErrorPageTransform.xsl en het bestand dat getransformeerd moet worden, een error page.

Het eerste blok dat uitgelezen wordt is het Description Block. De template om dat specifieke block te matchen is als volgt:

```
<xsl:template match="block[contains(label, 'Description')]">
   <descriptionBlock>
       <xsl:apply-templates select="para" />
   </descriptionBlock
</xsl:template>
```

Het description block in de XML bron zit als volgt in elkaar:

```
<block>
   <label>Description</label>
   <para maxOccurence=”unbounded”>
       <ptxt>Text line</ptxt>
   </para>
</block>
```

Dus, om <para> te transformeren naar <textLine> hebben we deze template gemaakt:

```
<xsl:template match="para[ptxt]">
  <xsl:for-each select="ptxt">
      <textLine>
          <xsl:value-of select="." />
```

```
    </textLine>
  </xsl:for-each>
</xsl:template>
```

Wat er eigenlijk gebeurd is dit: de <para> tag wordt er tussenuit gegooid en de <ptxt> tag wordt naar een<textLine> tag veranderd.
Er zijn daarnaast ook nog <para> element zonder een ptxt element. Deze worden zoals hieronder afgehandeld

```
<xsl:template match="para" />
```

Oftewel, er wordt niks mee gedaan.

Nadat het description blok is getransformeerd zou het test & check block getransformeerd moeten worden. Het is alleen onmogelijk gebleken om dit te doen in de tijd die we hadden. Wij besloten onze aandacht eerst te richten op het transformeren van het Suspected Parts & Settings block.

Deze wordt als volgt getransformeerd:

In de bron XML staan de Suspected Parts & Setting als volgt weergegeven:

```
<block>
   <label>Suspected Parts & Settings</label>
   <table>
       <theader>
       <tbody>
        <row maxOccurance='unbounded'>
         <entry colName='col1 to col4 and col7'>
    </row>
        </tbody>
   </table>
</block>
```

De row nodes staan in feite gelijk aan suspect nodes. Waarbij elke entry node een aantal verschillende eigenschappen hebben. Naast andere attributen heeft elke entry node een colName (col1, col2, col3, col4 en col7). Deze colNames geven indirect aan wat de content inhoudt die binnen dat kolom staat. Col1 bevat reference naar een parts list, col2 bevat de description, col3 links naar SDS tests, col4 'actions', waar in feite niet veel mee gedaan wordt en col7 bevat links naar electrical diagrams.

Een row node wordt in een aantal stappen getransformeerd. Eerst wordt in de error page XML een <suspect> element aangemaakt, met een referentie naar een part of setting en wat voor type suspect het is. Daar wordt deze code voor gebruikt:

```
<suspect>
  <xsl:attribute name="type">
       <xsl:text>part</xsl:text>
  </xsl:attribute>
  <xsl:for-each select="entry[@colname='col2']">
       <xsl:value-of select="normalize-space(ptxt[1])" />
  </xsl:for-each>
</suspect>
```

Daarna worden alle entry nodes nog een keer doorlopen en getransformeerd. Dit alles wordt dan naar een nieuwe outputfile geschreven, de zogenaamde part page of setting page.
Elke entry node wordt aan de hand van z'n colname op een andere manier getransformeerd.

Col1:

```
<xsl:when test="@colname='col1'">
    <xsl:variable name="docstring" select="concat($moduleLoc, ./ptxt[1]/xref/@href, '.xml')" />
   <xsl:variable name="indexNr" select="(tokenize(ptxt[1]/xref,'-'))[last()]" />
   <partListImage>
       <xsl:attribute name="name">
           <xsl:value-of select="ptxt/xref" />
       </xsl:attribute>
       <xsl:attribute name="index">
           <xsl:value-of select="$indexNr" />
       </xsl:attribute>
       <xsl:attribute name="src">
           <xsl:value-of select="substring-after(document($docstring)/module/block[contains(label,
     'Illustration')]/figure/sheet/@href,'\')" />
       </xsl:attribute>
   </partListImage>
   <descriptionBlock>
       <textLine>
           <xsl:value-of select="document($docstring)/module/blockblock[contains(label,
     'Index')]/item[@posno =$indexNr]/description" />
       </textLine>
   </descriptionBlock>
   <partID>
       <xsl:value-of select="document($docstring)/module/block[contains(label,
     'Index')]/item[@posno =$indexNr]/partno" />
   </partID>
</xsl:when>
```

In deze XSL code wordt er, alleen voor parts, een partListImage element, inclusief attributen, aangemaakt. Daarnaast wordt er een descriptionBlock en partID uit de gerefereerde partslist gehaald.

Col2:

```
<xsl:when test="@colname='col2'">
  <title>
       <xsl:apply-templates select="ptxt[not(xref)]" />
  </title>
</xsl:when>
```

Uit col2 wordt voor elk suspect de title gehaald. Deze staat opgeslagen in een ptxt die op deze manier verwerkt wordt:

```
<xsl:template match="ptxt[not(xref) and not(uitext)]">
  <xsl:value-of select="." />
</xsl:template>
```

Oftewel, alleen de text value wordt uit het element gehaald.

Col3:

Col3 bestaat uit links naar SDS tests en wordt op deze manier in de XSL verwerkt:

```
<xsl:when test="@colname='col3'">
  <SDStests>
     <xsl:for-each select="ptxt[.!='']">
         <SDStest>
             <xsl:apply-templates select="." />
         </SDStest>
     </xsl:for-each>
  </SDStests>
</xsl:when>
```

Het kan zijn dat er meerdere ptxt elementen (oftewel meerdere SDStests) in een kolom beschreven staan, deze worden elk uitgelezen en in de SDStests lijst gezet.
Over het algemeen matchen de ptxt elementen binnen een col3 entry node met deze template:

```
<xsl:template match="ptxt[xref]">
  <xsl:attribute name="moduleNr">
       <xsl:value-of select="xref/@href" />
  </xsl:attribute>
  <xsl:attribute name="name">
       <xsl:value-of select="xref" />
  </xsl:attribute>
</xsl:template>
```

Er wordt een <SDStest> element gemaakt, en hier worden de nodige attributen ingehangen, uitgelezen uit het xref element binnen het ptxt element.

Col4:

```
<xsl:when test="@colname='col4'">
  <actionList>
     <xsl:for-each select="ptxt">
         <xsl:choose>
              <xsl:when test="xref">
                  <actionItem>
                      <xsl:apply-templates select="." />
                  </actionItem>
              </xsl:when>
              <xsl:otherwise>
                  <actionString>
                      <xsl:apply-templates select="." />
                  </actionString>
              </xsl:otherwise>
          </xsl:choose>
       </xsl:for-each>
   </actionList>
</xsl:when>
```

Uit col4 wordt de actionList gehaald. Als een ptxt een xref element bevat, maken we hier een actionItem van. Zo niet, dan wordt het een actionString. ptxt[xref] wordt behandeld zoals in col3. Al het andere zoals ptxt in col2.

Col7:

```
<xsl:when test="@colname='col7'">
   <ElectricalDiagrams>
       <xsl:for-each select="ptxt[xref]">
           <link>
               <xsl:variable name="docstring2" select="concat($moduleLoc, xref/@href, '.xml')" />
               <xsl:attribute name="name">
                   <xsl:value-of select="xref" />
               </xsl:attribute>
               <xsl:attribute name="fileName">
                   <xsl:value-of select="document($docstring2)/module/title" />
               </xsl:attribute>
               <xsl:attribute name="src">
                   <xsl:value-of select="substring-
    after(document($docstring2)/module/block[contains(label,
    'Illustration')]/figure/sheet/@href,'\')" />
               </xsl:attribute>
           </link>
       </xsl:for-each>
   </ElectricalDiagrams>
</xsl:when>
```

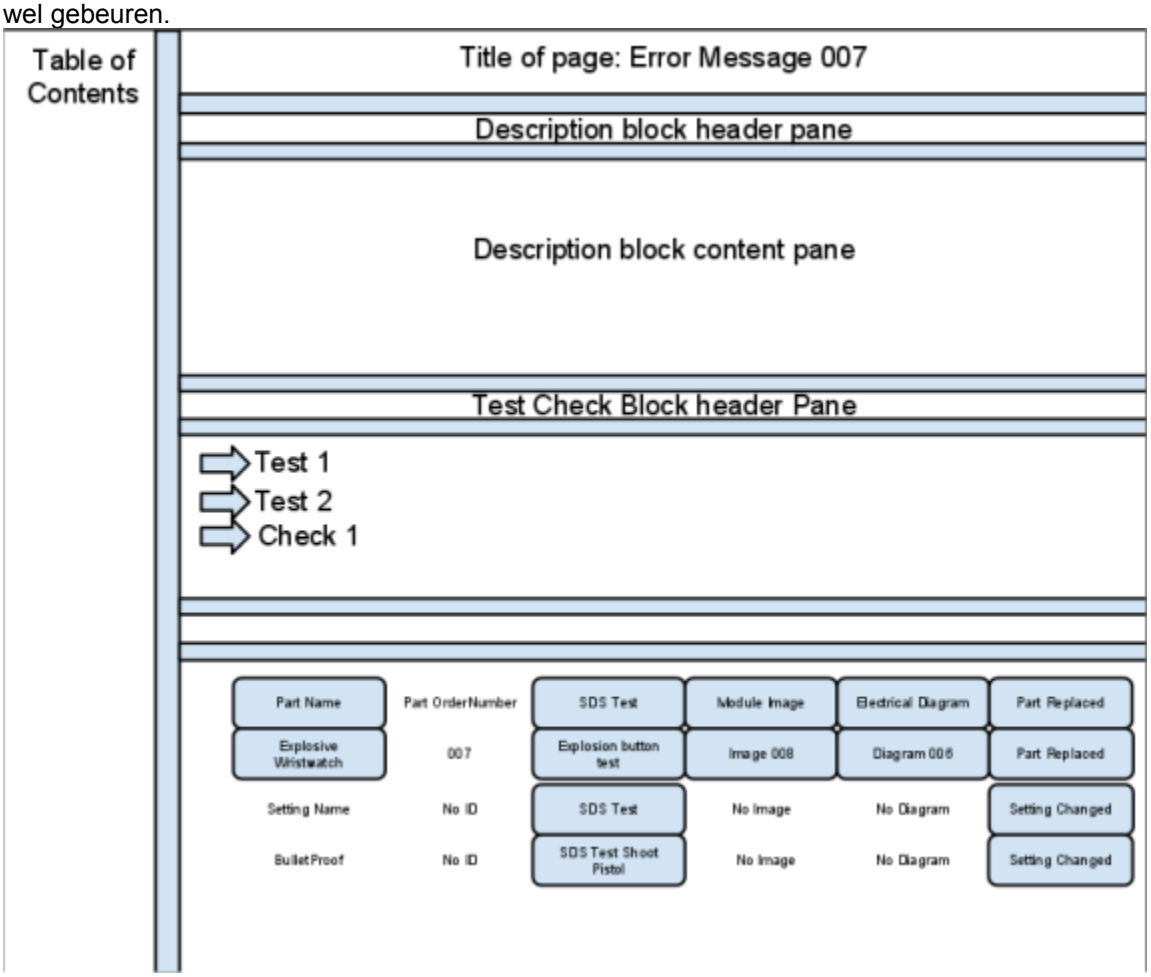
In col7 staan de referenties naar de electrical diagrams. Hier zijn er af en toe meerdere van te vinden in 1

kolom, dus voor deze entry wordt elke ptxt element dat een xref bevat uitgelezen. Daarna wordt het <link> element gemaakt, waarbij de src van de illustratie uit een andere bestand gehaald wordt; docstring2 geeft de locatie aan van dat bestand.

Nadat elke row node in het Suspected Parts & Settings Block is getransformeerd, is de volledige transformatie klaar. De output is: een error page volgens onze xsd, en voor elke suspect een part/setting page volgens onze xsd.

Hoofdstuk 4 - GUI

In Figuur 3 staat een algemene indruk van onze GUI. Aangezien de GUI in het systeem op een aantal plaatsen nog niet af is, is het nog niet mogelijk om een screenshot te plaatsen. Dit zal in het eindverslag wel gebeuren.



Figuur 3: GUI Detailed Error Page

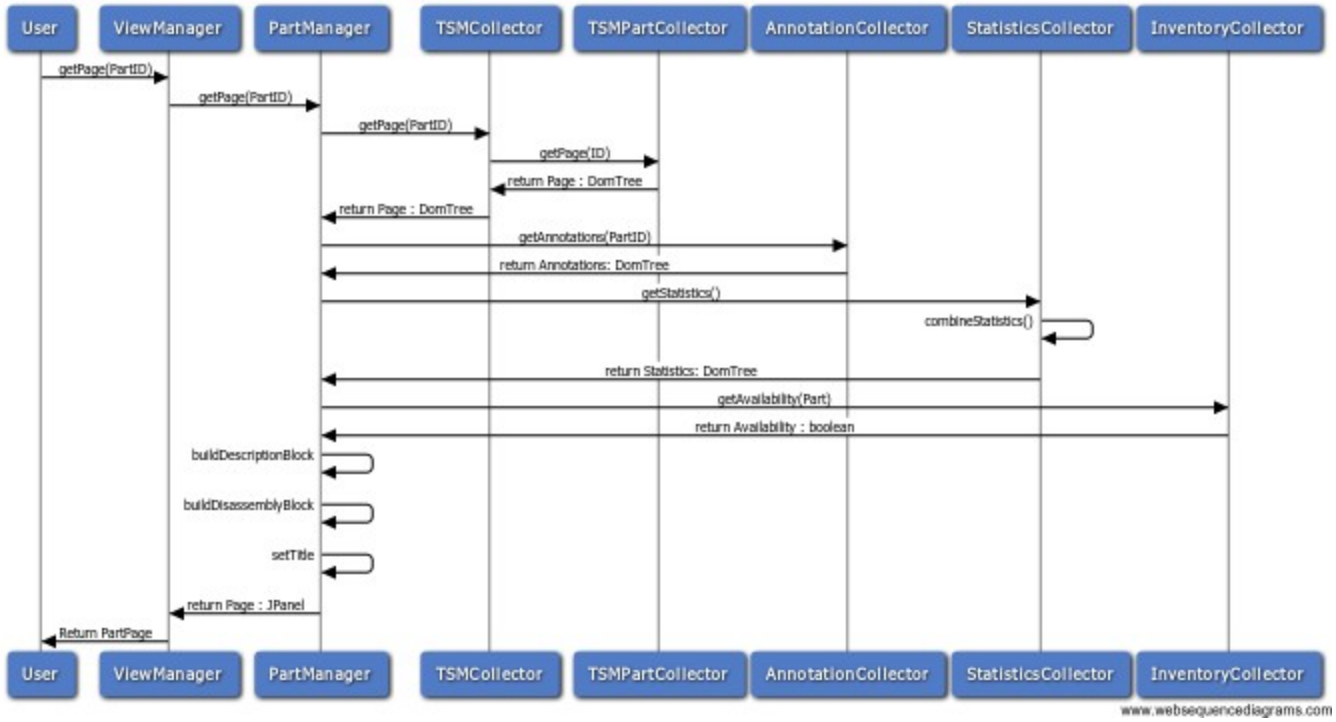
In ieder geval is de algemene lay-out van de GUI goed te zien. Het Table of Contents paneel aan de linkerkant is altijd te zien, zodat de gebruiker op elk moment via de ToC naar een pagina kan springen. De error page is hier opgebouwd uit 3 verschillende blokken (blockPanels), namelijk Description, Test&Check en Suspected Parts & Settings. De description is altijd een veld met tekst. Het Test & Check Block is een veld met een opsomming van alle tests en checks die mogelijk zijn. Als 1 van de tests aangeklikt wordt zal er via een dropdown veld een stappenplan te zien zijn, die de gebruiker stap voor stap door de tests en checks leidt. Aan het einde van dit stappenplan kan aangeven worden of de check gefaald heeft of niet, deze ‘feedback’ wordt dan verwerkt in de Suspected Parts & Settings block. Dit Block is eigenlijk een tabel, met alle parts en settings die voor de bepaalde error code kunnen zorgen (suspect zijn). De volgorde is gebaseerd op statistiek over hoe vaak een part kapot gaat in relatie tot een error code. Elke test en elke check is verbonden aan 1 of meer parts. De feedback die verkregen wordt bij het uitvoeren van de tests en checks kan verwerkt worden in de volgorde van de tabel. Dit betekent dat de monteur dankzij de aangepaste volgorde sneller de ‘suspect’ kan kiezen die waarschijnlijk verantwoordelijk is voor de error.

Hoofdstuk 5 - Sequence Diagrammen

In dit hoofdstuk bespreken we de Sequence Diagrammen van de functionaliteiten die op het moment van schrijven geïmplementeerd zijn. De onderstaande sequence diagrammen beschrijven welke aanroepen er gebeuren wanneer een bepaalde pagina aangemaakt moet worden. Hierbij zijn een aantal onderdelen niet weergegeven, aangezien deze het plaatje nodeloos groter maken en deze functionaliteiten nog niet aan de orde zijn om te implementeren.

5.1 Sequence Diagram voor Part Page

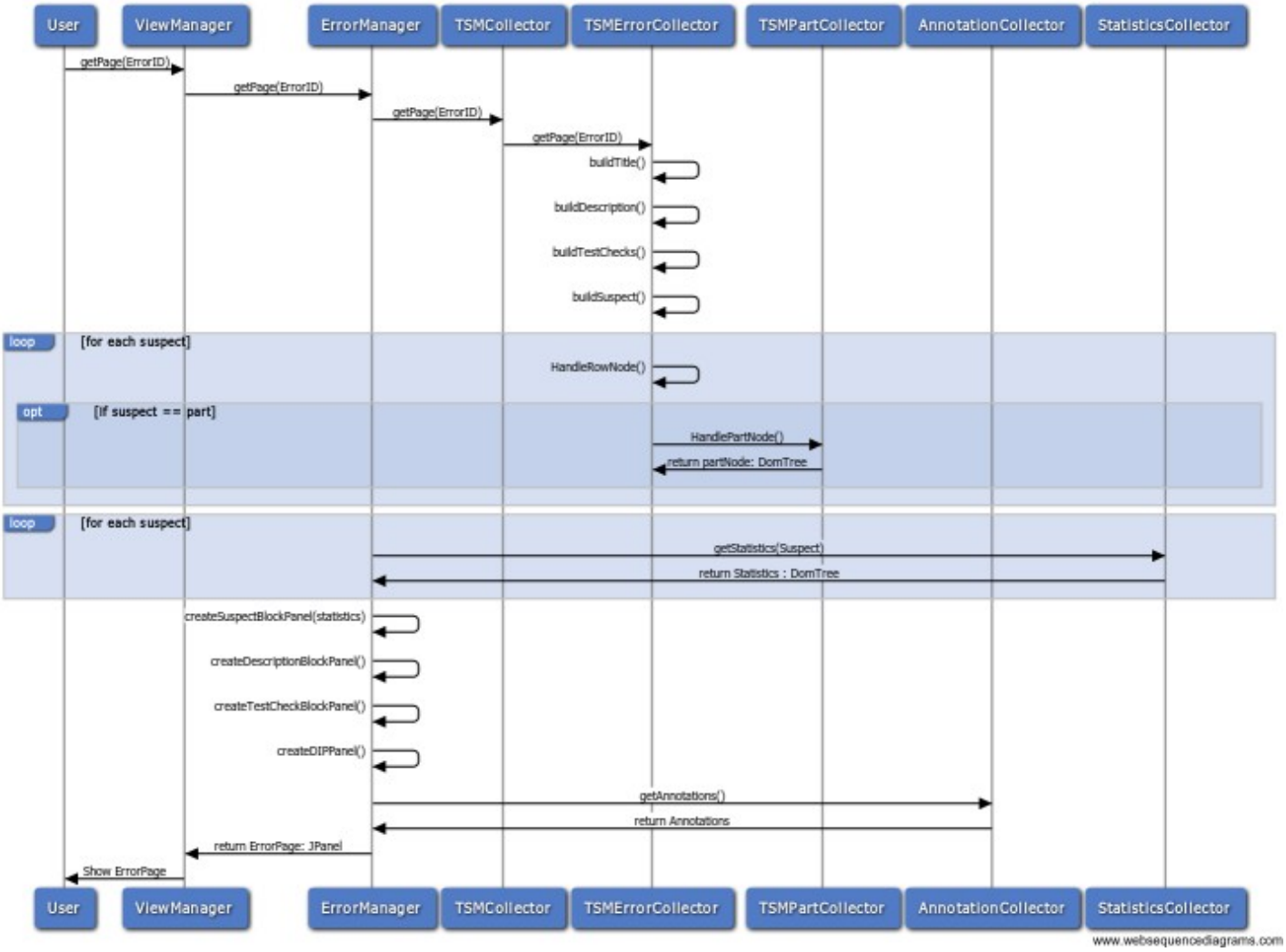
Figuur 4 bevat de aanroepen die nodig zijn om een Part Page op te bouwen. Vanaf het begin dat de User de part-pagina aanklikt tot het moment waarop de pagina weergegeven wordt.



Figuur 4: Sequence Diagram Parts Page

5.2 Sequence Diagram voor Error Page

De opzet van een Error Pagina is complexer dan het opzet van een Part Page. Dit komt onder andere door de blokken “Test & Checks” en “Suspected Parts & Settings”. Deze blokken moeten uit een aparte bron gehaald worden en allemaal geordend worden aan de hand van de statistieken die opgevraagd worden. Het sequence diagram wordt weergegeven in Figuur 5.



Figuur 5: Sequence Diagram Error Page

Appendix E – XSD Files

XSD Error Page

```
<?xml version="1.0" encoding="windows-1250"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<!-- <type> Basic element describing what type a suspect is, can only be errorCode for this page
-->
    <xs:element name="type" type="xs:string"/>
    <!-- <title> Basic element describing title of errorCode page -->
    <xs:element name="title" type="xs:string"/>
    <!-- <type> attribute containing type of suspect, can be part or setting -->
    <xs:attribute name="type" type="xs:string"/>
    <!-- <moduleNr> attribute containing moduleNr reference of a link -->
    <xs:attribute name="moduleNr" type="xs:string"/>
    <!--  <name> attribute containing the name of linked page -->
    <xs:attribute name="name" type="xs:string"/>
    <!--  <src> attribute containing pathname of (most likely) an image -->
    <xs:attribute name="src" type="xs:string"/>
    <!-- <fileName> attribute containing the name of a linked file. -->
    <xs:attribute name="fileName" type="xs:string"/>


    <xs:element name="module">
       <xs:complexType>
              <xs:sequence>
                     <xs:any processContents="strict" maxOccurs="unbounded"/>
              </xs:sequence>
       </xs:complexType>
    </xs:element>
    <!-- <descriptionBlock> block containing the description of an error code page.  -->
    <xs:element name="descriptionBlock">
       <xs:complexType>
              <xs:sequence>
                     <xs:element name="textLine" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
              </xs:sequence>
       </xs:complexType>
    </xs:element>
    <!-- <suspectBlock> block containing a list of suspects -->
    <xs:element name="suspectBlock">
       <xs:complexType>
              <xs:sequence>
                     <xs:element ref="suspect" minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
       </xs:complexType>
    </xs:element>
    <!-- element that contains a link to a part or setting. A suspect is suspect of causing the
error -->
    <xs:element name="suspect">
       <xs:complexType>
              <xs:simpleContent>
                     <xs:extension base="xs:string">
                     <xs:attribute name="type" type="xs:string"/>
                     </xs:extension>
              </xs:simpleContent>
       </xs:complexType>
    </xs:element>
</xs:schema>
```

XSD: Part Page

```
<?xml version="1.0" encoding="windows-1250"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<!-- <type> Basic element describing what type a suspect is, can only be part for this
page -->
    <xs:element name="type" type="xs:string"/>
	 <!-- <title> Basic element describing title of part page -->
    <xs:element name="title" type="xs:string"/>
    <!--  <partID> nonNegativeInteger element containing the ordernumber of a part -->
	<xs:element name="partID" type="xs:nonNegativeInteger" />
	<!--< actionString> contains a string gotten from the action column -->
	<xs:element name="actionString" type="xs:string" />
	<!--  <debugLine> contains a string with a debug message-->
		<xs:element name="debugLine" type="xs:string" />

	<!--  <name> attribute containing the name of linked page -->
    <xs:attribute name="name" type="xs:string"/>
	<!--TODO:create regex for attribute src-->
	 <!--  <src> attribute containing pathname of (most likely) an image -->
    <xs:attribute name="src" type="xs:string"/>
    <!-- <index> index contains the index number of this part in a partList -->
	<xs:attribute name="index" type="xs:nonNegativeInteger" />
	<!-- <moduleNr> attribute containing moduleNr reference of a link -->
    <xs:attribute name="moduleNr" type="xs:string"/>
	<!-- <fileName> attribute containing the name of a linked file. -->
    <xs:attribute name="fileName" type="xs:string"/>
	<xs:element name="module">
		<xs:complexType>
			<xs:sequence>
				<xs:any processContents="strict" maxOccurs="unbounded" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<!-- <partListImage> contains the link to the illustration of the module the part resides
in. -->
	<xs:element name="partListImage">
		<xs:complexType>
			<xs:attribute ref="name" />
			<xs:attribute ref="src" />
			<xs:attribute ref="index" />
		</xs:complexType>
	</xs:element>
	<!-- <descriptionBlock> block containing the description of a part.  -->
    <xs:element name="descriptionBlock">
		<xs:complexType>
			<xs:sequence>
				<xs:element name="textLine" type="xs:string"
maxOccurs="unbounded" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<!-- <SDStests> Element containing all the SDS tests available for part  -->
	<xs:element name="SDStests">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="SDStest" minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<!-- <SDStest> element containing name and link of a SDS test -->
	<xs:element name="SDStest">
		<xs:complexType>
```

```
                        <xs:attribute ref="moduleNr" />
                        <xs:attribute ref="name" />
                </xs:complexType>
        </xs:element>
        <!-- <actionList> element containing actionStrings or actionItems. Not very clear what to
do with this yet (probably get removed) -->
        <xs:element name="actionList">
                <xs:complexType>
                        <xs:sequence>
                                <xs:any minOccurs="0" maxOccurs="unbounded"
processContents="strict"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <!--  <actionItem> element containing a reference to (most likely, but unsure) a dis-
assembly page. -->
        <xs:element name="actionItem">
                <xs:complexType>
                        <xs:attribute ref="moduleNr" />
                        <xs:attribute ref="name" />
                </xs:complexType>
        </xs:element>
        <!-- <ElectricalDiagrams> element containing all links to electrical Diagrams necessary
for the part -->
        <xs:element name="ElectricalDiagrams">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="link" minOccurs="0" maxOccurs="unbounded" />
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <!-- <link> reference to electricalDiagram (an illustration) -->
        <xs:element name="link">
                <xs:complexType>
                        <xs:attribute ref="name" />
                        <xs:attribute ref="src" />
                        <xs:attribute ref="fileName" />
                </xs:complexType>
        </xs:element>
</xs:schema>
```

XSD: Setting Page

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

        <xs:element name="settingList" type="xs:string" />
        <!-- <type> Basic element describing what type a suspect is, can only be
                part for this page -->
        <xs:element name="type" type="xs:string" />
        <!-- <title> Basic element describing title of setting page -->
        <xs:element name="title" type="xs:string" />
        <!--< actionString> contains a string gotten from the action column -->
        <xs:element name="actionString" type="xs:string" />
        <!-- <debugLine> contains a string with a debug message -->
        <xs:element name="debugLine" type="xs:string" />

        <!-- <name> attribute containing the name of linked page -->
        <xs:attribute name="name" type="xs:string" />
        <!--TODO:create regex for attribute src -->
        <!-- <src> attribute containing pathname of (most likely) an image -->
        <xs:attribute name="src" type="xs:string" />
```

```
	<!-- <index> index contains the index number of this part in a partList -->
	<xs:attribute name="index" type="xs:nonNegativeInteger" />
	<!-- <moduleNr> attribute containing moduleNr reference of a link -->
	<xs:attribute name="moduleNr" type="xs:string" />
	<!-- <fileName> attribute containing the name of a linked file. -->
	<xs:attribute name="fileName" type="xs:string" />
	<xs:element name="module">
		<xs:complexType>
			<xs:sequence>
				<xs:any processContents="strict" maxOccurs="unbounded" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>

	<!-- <SDStests> Element containing all the SDS tests available for part -->
	<xs:element name="SDStests">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="SDStest" minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<!-- <SDStest> element containing name and link of a SDS test -->
	<xs:element name="SDStest">
		<xs:complexType>
			<xs:attribute ref="moduleNr" />
			<xs:attribute ref="name" />
		</xs:complexType>
	</xs:element>
	<!-- <actionList> element containing actionStrings or actionItems. Not very
		clear what to do with this yet (probably get removed) -->
	<xs:element name="actionList">
		<xs:complexType>
			<xs:sequence>
				<xs:any minOccurs="0" maxOccurs="unbounded"
processContents="strict" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<!-- <actionItem> element containing a reference to (most likely, but unsure)
		a dis-assembly page. -->
	<xs:element name="actionItem">
		<xs:complexType>
			<xs:attribute ref="moduleNr" />
			<xs:attribute ref="name" />
		</xs:complexType>
	</xs:element>
	<!-- <ElectricalDiagrams> element containing all links to electrical Diagrams
		necessary for the setting -->
	<xs:element name="ElectricalDiagrams">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="link" minOccurs="0" maxOccurs="unbounded" />
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<!-- <link> reference to electricalDiagram (an illustration) -->
	<xs:element name="link">
		<xs:complexType>
			<xs:attribute ref="name" />
			<xs:attribute ref="src" />
			<xs:attribute ref="fileName" />
```

```
                </xs:complexType>
        </xs:element>

</xs:schema>
```

Appendix F – Code Review SIG

Eerste Review:
De code van het systeem scoort bijna 5 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De reden dat de perfecte score van 5 sterren niet wordt gehaald zijn de lagere scores voor Unit Size en Unit Complexity.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is en daardoor eenvoudiger te onderhouden wordt. Binnen de bovengemiddeld lange methodes in dit systeem, zoals bijvoorbeeld de 'handlePartEle'-methode in 'TSMSuspectBlockPanel.java', zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Commentaarregels zoals bijvoorbeeld '//Create link to SDS Tests' of '//Creates a link to the electrical diagram of the button' geven altijd een goede indicatie dat er een autonoom stuk functionaliteit te ontdekken is. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Op eenzelfde manier word er voor Unit Complexity gekeken naar het percentage code wat bovengemiddeld lang is. Ook hier geldt dat het opsplitsen van dit soort methodes in kleinere stukken ervoor zorgt dat elk onderdeel makkelijker te begrijpen, te testen en daardoor te onderhouden wordt. In dit geval komen de meest complexe methoden ook naar voren als de langste methoden, waardoor het oplossen van het eerste probleem ook dit probleem zal verhelpen.

Wat verder opvalt is de aanwezigheid van een tweetal test-applicaties, 'ReadAllErrorPagesTest.java' en 'XMLParserTest.java'. Het is goed om te zien dat er gebruik wordt gemaakt van een vorm van automatisch testen. Wat wel opvalt binnen deze applicaties is dat ook hier complexere methodes inzitten (bijvoorbeeld de methode 'handleRowNode'). Het is aan te raden de test-code zo simpel mogelijk te houden, het inzetten van unit-test kan daarbij helpen. De aanwezigheid van een directory genaamd 'JUnitTest' is veelbelovend, hopelijk wordt deze in de rest van het traject ook gevuld.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden zodra het systeem verder groeit.

Tweede Review:
In de tweede upload zien we dat zowel de omvang van het systeem als de score voor onderhoudbaarheid is gestegen. Op het gebied van de lengte van methodes is een flinke vooruitgang geboekt, er zijn geen bovengemiddeld lange methodes meer te vinden in het systeem. Daarnaast is ook de complexiteit van de methodes gedaald, in deze upload is alle code gedefinieerd in methodes met een lage complexiteit. De test-applicaties 'ReadAllErrorPagesTest.java' en 'XMLParserTest.java' zijn niet meer terug te vinden in de upload, de test-applicatie 'ErrorPageParserTest.java' lijkt hiervoor in de plaats te zijn gekomen. Deze laatste applicatie geeft hetzelfde beeld als de eerdere test-applicaties, ook hierin zijn (te) complexe methodes te vinden.

Uit deze observaties kunnen we concluderen dat de meeste aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject. Het is goed om te zien dat de studenten de bovengemiddelde score vast hebben weten te houden.