Technische Universiteit Delft
Tam Tam

# Knowledge Network

Eindverslag Bachelorproject

Joost-Wim Boekesteijn (1174355)
Benjamin W. Broersma (1174401)

Bachelorproject (IN3700) Technische Informatica
Faculteit Elektrotechniek, Wiskunde en Informatica

Commissie:
ir. Bart Manuel (opdrachtgever Tam Tam)
ir. Hans Geers (stagebegeleider TU Delft)
ir. Bernard Sodoyer (stagecoördinator TU Delft)

2007-11-26
V. 1.0

# 1 Voorwoord

In de maanden juli t/m oktober hebben we bij Tam Tam gewerkt aan ons bachelorproject. Dit project is het afsluitende onderdeel van het bachelorprogramma Technische Informatica aan de Technische Universiteit Delft. In de genoemde periode hebben we 12 weken lang in het kantoor van Tam Tam te Rijswijk gewerkt. De overige weken hebben we gebruikt voor het voorbereiden van tentamens.

We willen Bart Manuel, Wouter Geurtzen en Thijs ter Beek bedanken voor de begeleiding van ons project vanuit Tam Tam. Verder willen we Hans Geers bedanken voor de begeleiding vanuit de TU Delft en de hulp bij het schrijven van de documenten. Daarnaast bedanken we de collega's bij Tam Tam voor de prettige sfeer en de tijd die ze hebben genomen om vragen te beantwoorden.

Joost-Wim Boekesteijn
Benjamin W. Broersma

Delft, november 2007

# Inhoudsopgave

# 2 Samenvatting

Binnen Tam Tam is kennis van de medewerkers verspreid opgeslagen in verschillende informatiesystemen. Het doel van het bedrijf is dat al deze kennis vanuit een centraal punt effectief kan worden doorzocht. Dit centrale punt is het SharePoint-platform, een product van Microsoft. Voor dit project is SharePoint uitgebreid zodat het mogelijk wordt om alle informatie te voorzien van tags en ratings. Gebruikers kunnen deze extra metadata gebruiken bij het zoeken in het systeem.

Aan het begin van dit project is er een analyse gemaakt van de huidige systemen die binnen Tam Tam worden gebruikt om informatie te delen. Aan de hand van de resultaten van deze analyse is er een ontwerp gemaakt voor de uitbreiding op SharePoint. Dit ontwerp is geïmplementeerd en getest in de SharePoint-omgeving. Uiteindelijk is er binnen 12 weken een werkend prototype opgeleverd, dat door eindgebruikers is getest.

Wij hadden voor de start van dit project geen ervaring met het SharePoint-platform. Aan het begin van het project is er daarom veel onderzoek gedaan door het schrijven van kleine tests. Dit onderzoek werd bemoeilijkt doordat bepaalde delen van het SharePoint-platform niet of onvolledig zijn gedocumenteerd. Delen van de geschreven tests zijn gebruikt in de uiteindelijke versie van ons prototype. Het ontwerp van sommige onderdelen van het systeem is hierdoor pas achteraf ontstaan.

# 3 Inleiding

In dit hoofdstuk wordt een beschrijving gegeven van Tam Tam en wordt uitgelegd welk probleem er voor dit project is opgelost.

## 3.1 Tam Tam

Tam Tam is een full-service internetbedrijf en is gespecialiseerd in portals, business applicaties, marketingcommunicatie-sites en online network marketing. Op dit moment heeft Tam Tam ongeveer 90 medewerkers, verspreid over twee kantoren in Rijswijk en Utrecht.

Binnen Tam Tam is er behoefte om de kennis van medewerkers met elkaar te delen. In het verleden werd dit gedaan met public folders op een Microsoft Exchange Server, waar niet goed in kon worden gezocht. Dit medium wordt niet meer actief gebruikt maar bevat wel veel informatie. Op dit moment heeft een aantal medewerkers een Tam Tam weblog, deze weblogs zijn toegankelijk voor de buitenwereld. Hiernaast zijn er nog interne mailinglijsten die gebruikt worden voor kennisoverdracht, deze worden actief gebruikt. Binnen Tam Tam wordt Microsoft Office SharePoint Server 2007 gebruikt als primair platform voor de ondersteuning van de bedrijfsprocessen. Het delen en doorzoekbaar maken van informatie is hier een belangrijk onderdeel van. Tam Tam loopt in Nederland voorop in het gebruik van SharePoint. Voor klantoplossingen wordt bij voorkeur onderzocht hoe SharePoint hiervoor ingezet kan worden.

## 3.2 Probleemstelling

Vanwege de leidende rol die Tam Tam heeft op het gebied van SharePoint-implementaties voor klanten, wordt dit platform ook binnen het bedrijf ingezet voor het delen en doorzoekbaar maken van informatie. Het is gewenst dat zo veel mogelijk van de hierboven genoemde systemen (Exchange public folders, weblogs, mailinglijsten) binnen SharePoint toegankelijk en doorzoekbaar worden gemaakt.

# 4 Analyse

Voor de analyse van het probleem, waarvoor we de eerste weken van het project hebben gebruikt, zijn er interviews en brainstormsessies gehouden met medewerkers van Tam Tam. We hebben onderzocht welke systemen er op dit moment worden gebruikt voor het delen van informatie binnen het bedrijf. Daarna is er een lijst van functionele en niet-functionele eisen opgesteld waaraan het prototype zal moeten voldoen. Tenslotte zijn er systeemmodellen gemaakt, waarvoor we gebruik hebben gemaakt van scenario's en use cases.

## 4.1 Werkwijze

Voor het verzamelen van de eisen van het systeem is er gebruik gemaakt van verschillende methoden. We waren niet goed bekend zijn met de precieze mogelijkheden van het SharePoint-platform en wisten daarom ook niet zeker of al onze ideeën technisch haalbaar zijn. We wilden tijdens de loop van het project de mogelijkheid hebben aanpassingen te maken door nieuwe inzichten van ons of gewijzigde wensen van Tam Tam. Daarom maakten we gebruik van technieken uit het Agile software development model. Er is gebruik gemaakt van brainstormsessies, interviews, gesprekken met experts en gesprekken met gebruikers. Ook is gekozen voor prototyping om snel feedback te krijgen en het project te kunnen bijsturen.

## 4.2 Huidig systeem

Op basis van de informatie uit gesprekken met medewerkers hebben we een inventarisatie gemaakt van de verschillende informatiebronnen in het bedrijf en de manier waarop deze worden gebruikt. We hebben de volgende informatiebronnen gevonden:

**Public Folders**

Public folders zijn mappen met e-mailberichten die door meerdere mensen kunnen worden gelezen. Deze mappen werden een paar jaar geleden nog veel gebruikt. Op dit moment worden er nauwelijks nog berichten in geplaatst. De public folders kunnen vanuit Outlook worden doorzocht, maar het is niet mogelijk om er vanuit de SharePoint in te zoeken vanwege een configuratiefout van de zoeksoftware. Vroeger was dit wel mogelijk.

**Weblogs**

Vanaf oktober 2003 worden er door medewerkers van Tam Tam weblogs bijgehouden die bedoeld zijn voor externe communicatie. Elk weblog heeft een eigen zoekfunctie waarmee kan worden gezocht in de tekst van de berichten. Het is niet mogelijk om vanuit een centraal punt in alle weblogs te zoeken.

**Mailinglijsten**

Voor allerlei soorten interne mededelingen naar alle werknemers of bepaalde groepen medewerkers binnen het bedrijf worden mailinglijsten gebruikt. Er is geen centraal archief waarin deze berichten worden bewaard, medewerkers kunnen enkel zoeken in de berichten die ze zelf hebben ontvangen.

**SharePoint**

Het doel van Tam Tam is alle bedrijfsinformatie via SharePoint toegankelijk en doorzoekbaar te maken. Op dit moment wordt er voor elk nieuw project een Project Space aangemaakt met daarin een documentbibliotheek waarin medewerkers projectdocumentatie plaatsen. Een documentbibliotheek ondersteunt versiebeheer en de documenten die hier staan worden geïndexeerd, zodat er op tekst in de documenten kan worden gezocht.

Naast de Project Spaces wordt er tegenwoordig ook informatie in wiki's geplaatst. Deze wiki's bevatten algemene informatie en zijn voor iedereen toegankelijk. De inhoud van de wiki-pagina's wordt door SharePoint geïndexeerd.

SharePoint heeft een uitgebreid systeem voor het indexeren van gegevens. Binnen Tam Tam is er extra software geïnstalleerd waarmee het mogelijk is om zoekresultaten te filteren op zelf in te stellen metadata.

Op basis van de verschillende informatiebronnen en de manier waarop deze worden gebruikt, hebben we in overleg met de opdrachtgever besloten een uitbreiding op SharePoint te maken waardoor er aan documenten in SharePoint tags en ratings kunnen worden toegekend. Bij het zoeken naar informatie wordt deze metadata gebruikt om de zoekresultaten te filteren. Daarnaast zullen de weblogs worden geïndexeerd in SharePoint, zodat de weblogberichten centraal kunnen worden doorzocht. Als aanvulling op de functionaliteit van de huidige mailinglijsten en Public Folders zullen we het mogelijk maken artikelen in wiki's toe te voegen door een e-mailbericht te sturen.

## 4.3 Functionele eisen

Het te ontwikkelen systeem is een uitbreiding van SharePoint met metadata (rating, tags) waarop kan worden gezocht, gesorteerd en gefilterd. Bestaande gegevens zoals e-mailberichten in Public Folders, mailinglijsten en weblog-artikelen kunnen vanuit SharePoint worden doorzocht.

Om gebruikers effectiever te laten zoeken naar informatie hebben we besloten metadata toe te voegen aan de informatie die aanwezig is in SharePoint. De metadata bestaat uit tags en ratings die aan elk item in SharePoint kunnen worden toegevoegd. Deze metadata wordt vervolgens gebruikt om zoekresultaten te sorteren of te filteren. Hierdoor zullen gebruikers specifiekere zoekresultaten kunnen krijgen. De gebruiker heeft de volgende mogelijkheden:

1) Rating van een item bekijken.
2) Lijstitems sorteren op rating.
3) Rating geven aan een item.
4) Tag cloud van een lijst bekijken.
5) Tags van een item bekijken.
6) Tags toekennen aan een item.
7) Lijstitems filteren op tags.
8) Zoekresultaten sorteren op rating.
9) Zoekresultaten filteren op tags.
10) Zoekresultaten filteren op bron.

## 4.4 Systeemmodellen

In het Requirements Analysis Document zijn systeemmodellen zoals scenario's en use cases opgenomen. Hierin wordt in detail beschreven hoe de gebruiker alle taken uit de lijst van functionele eisen uitvoert en welke operaties er door ons systeem zullen worden uitgevoerd.

# 5 Ontwerp

Dit deel van het document beschrijft het ontwerp van het systeem waarmee aan SharePoint tag- en ratingfunctionaliteiten worden toegevoegd.

## 5.1 Ontwerpcriteria

Beslissingen in het ontwerp zijn op basis van de volgende criteria gemaakt (belangrijkste eerst): stabiliteit, veiligheid, gebruiksvriendelijkheid, schaalbaarheid en snelheid. Het prototype moet zo veel mogelijk voldoen aan deze eisen.

### 5.1.1 Stabiliteit

Een SharePoint pagina kan onbruikbaar worden als een component dat deel uitmaakt van een pagina een ongeldige bewerking uitvoert. SharePoint geeft de mogelijkheid componenten van een pagina uit te schakelen. Als een pagina niet werkt zullen veel gebruikers echter niet weten hoe ze dit kunnen oplossen. De componenten die geschreven zullen worden, moeten stabiel zijn en geen ongeldige bewerkingen uitvoeren zodat alle pagina's altijd beschikbaar zullen zijn.

### 5.1.2 Veiligheid

Het prototype wordt op basis van SharePoint ontwikkeld. SharePoint heeft een rechtensysteem waarin gebruikers en groepen gebruikers op lijst- of itemniveau een set van rechten krijgen. Informatie uit dit rechtensysteem moet worden gebruikt bij de ontwikkeling van het nieuwe systeem. Gebruikers mogen geen toegang krijgen tot data bij items waar ze geen rechten voor hebben.

### 5.1.3 Gebruiksvriendelijkheid

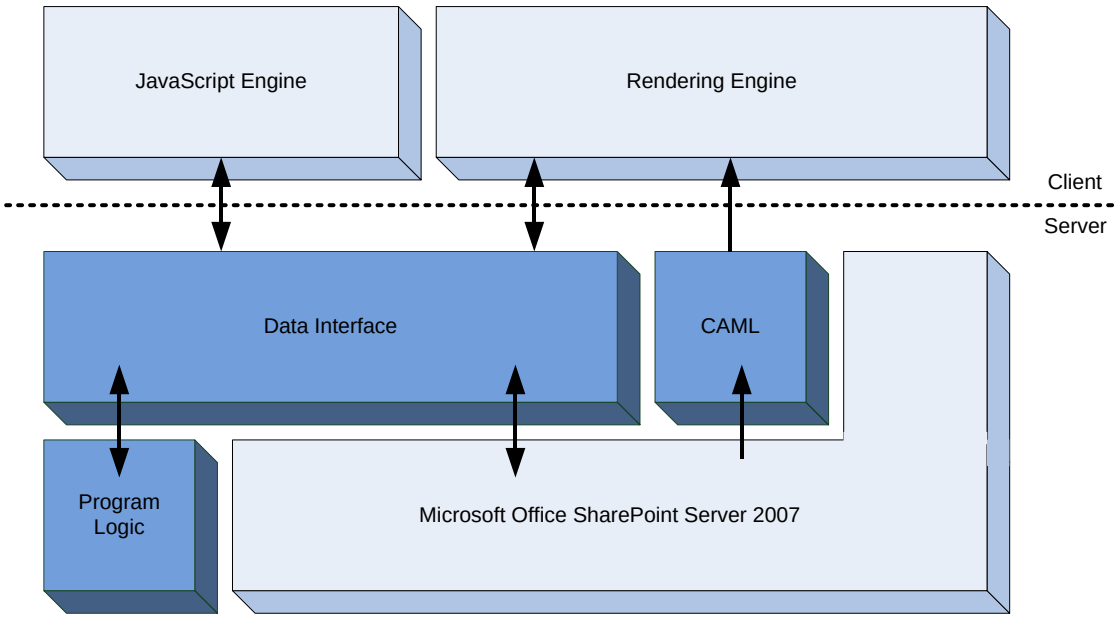
De bediening van het systeem moet voor zichzelf spreken. Het systeem moet zonder uitleg gebruikt kunnen worden, iedereen moet automatisch begrijpen hoe de bediening werkt.

### 5.1.4 Snelheid en schaalbaarheid

De tags en rating mogelijkheid moeten het gebruik van een lijst niet vertragen. Een lijst van 500 items met rating en tags moet niet meer dan 5 seconden trager worden weergegeven.

## 5.2 Decompositie in subsystemen

Het systeem wordt gebouwd in SharePoint en is gebaseerd op een webserver model. De verdeling in subsystemen van het te ontwikkelen systeem is gebaseerd op het Model-View-Controller model.




Figuur 1: Relaties tussen de subsystemen

### Model (Program Logic)

De programmalogica heeft geen verbindingen met SharePoint en kan los gebruikt worden, deze zal geïmplementeerd worden in het .NET Framework. Dit subsysteem bevat de basisklassen voor tags en rating. Voor deze klassen zijn unit tests geschreven.

### View (CAML, Data Interface)

Het schrijven van *CAML-View* code is in SharePoint nodig om een eigen weergave te maken voor een kolom, de uitvoer hiervan is HTML. Deze HTML data wordt verstuurd naar de webbrowser. Naast de code in CAML zijn er in de Data Interface WebParts/componenten voor het genereren van een HTML tagcloud.

### Controller (Data Interface)

De Data Interface is een koppeling tussen de webbrowser, de programmalogica en de opslag van SharePoint-data in lijsten. Op het moment dat de webbrowser een pagina opvraagt, zal de controller de programmalogica aanroepen en de uitvoer naar de webbrowser terugsturen.

In het Object Design zijn deze subsystemen verder uitgewerkt. In klassendiagrammen is de relatie te zien tussen de klassen in de subsystemen. Daarnaast is er van twee klassen een volledige specificatie gegeven met tekstuele beschrijving van alle methoden en properties.

# 6 Implementatie

Voor de implementatiefase van het project zijn de klassen uit het Object Design Document geïmplementeerd waarbij unit tests zijn geschreven voor onze eigen klassen met de functionaliteit voor tagging en rating. Bij het schrijven van de code hebben we ons gehouden aan de coding convention van Tam Tam, die is gebaseerd op de standaarden die Microsoft hiervoor publiceert. Details hierover zijn te vinden in het Object Design Document.

## 6.1 Tests

Tijdens het ontwikkelen van het systeem zijn er unit tests gemaakt voor de interne klassen. Deze tests controleren of de geschreven code zich houdt aan de specificaties.

De functionaliteit van het systeem wordt getest aan de hand van de use cases uit het Requirements Analysis Document. Dit houdt in dat gebruikers en beheerders de taken uit de use cases succesvol kunnen uitvoeren. Daarnaast worden er tests uitgevoerd die betrekking hebben op in het Architectural Design Document genoemde criteria als veiligheid, gebruiksvriendelijkheid en snelheid.

### 6.1.1 Statische code-analyse

Voor de code die is geschreven in C# en JavaScript is er gebruik gemaakt van statische analyse. Deze analyse is voor C# uitgevoerd met Microsoft FxCop[1] en Code Analysis van Microsoft Visual Studio 2005 Team Edition. De JavaScript-code is gecontroleerd met behulp van JSLint[2]. Statische analyse van code helpt om syntaxfouten te detecteren en kan een groot aantal 'bad practices' herkennen, maar kan niet controleren of de logica klopt en of klassen voldoen aan hun specificaties. Daarom zijn er op het niveau van de code ook unit tests uitgevoerd.

### 6.1.2 Unit tests

Om voor de klassen die de programmalogica implementeren te controleren of de methoden in de klassen werken zoals verwacht, worden er unit tests gebruikt. De unit tests worden met NUnit[3] uitgevoerd volgens de methode van test-driven development. Op basis van de specificaties van een klasse worden er tests geschreven voor die klasse. Wanneer dit is gedaan, zullen alle tests falen. Hierna wordt de klasse zelf geschreven en worden de tests opnieuw uitgevoerd. Pas wanneer alle tests succesvol worden uitgevoerd, voldoet de klasse aan de specificaties, mits de test consistent is met de specificatie. Na het maken van wijzigingen aan de code moeten de tests weer succesvol kunnen worden uitgevoerd (regressietests). Bij voorkeur worden de tests en de code voor de klasse door verschillende personen geschreven.

### 6.1.3 Integratietests

Tijdens de integratietests worden alle losse componenten gecombineerd en samen getest. Er is geen formele integratietest uitgevoerd in een aparte projectfase, omdat deze tests tijdens het ontwikkelen al zijn uitgevoerd. In dit project betekende het integreren dat de code binnen de SharePoint-omgeving werd uitgevoerd. Dit is tijdens het ontwikkelen vrijwel continu gedaan omdat van tevoren niet altijd bekend wat het gedrag van SharePoint zou zijn, vanwege ontbrekende of onvolledige documentatie. Door deze manier van werken hebben we besloten om geen aparte testfase te introduceren voor de integratietests.

---

[1] FxCop, http://www.gotdotnet.com/Team/FxCop/
[2] JsLint, http://www.jslint.com/lint.html
[3] NUnit, http://www.nunit.org/

### 6.1.4 Systeemtests

De systeemtests zijn uitgevoerd op het volledige systeem en niet meer op losse onderdelen zoals bij de unit tests. In het Architectural Design Document zijn veiligheid en snelheid/schaalbaarheid als ontwerpcriteria genoemd. Deze criteria zijn voor de systeemtests getest.

### 6.1.5 Gebruikersvriendelijkheidtests

De tests voor gebruiksvriendelijkheid worden uitgevoerd door eindgebruikers. Voor deze tests wordt 'hallway testing'[4] gebruikt, waarbij er door vijf willekeurig gekozen gebruikers[5] een serie van simpele taken zal worden uitgevoerd. De taken zijn gebaseerd op de scenario's, functionele eisen en use cases uit het Requirements Analysis Document. De lijst van taken is te vinden in bijlage A bij het Testrapport.

[4] Joel Spolsky, http://www.joelonsoftware.com/articles/fog0000000043.html
[5] Jakob Nielsen, http://www.useit.com/alertbox/20000319.html

# 7 Conclusies

Het gemaakte prototype voor SharePoint biedt de mogelijkheid om aan alle lijstitems een rating en tags te geven. Hierop kan ook gefilterd worden op lijstniveau. Ook is er integratie gemaakt met de SharePoint Search, er kan op rating gesorteerd worden en op tags gezocht worden. Er kan email gestuurd worden naar wiki's en er is een configuratiebestand gemaakt voor het indexeren van de weblogs. Aan twee eisen hebben we niet kunnen voldoen: het aanbieden van een contextafhankelijke tagcloud op de pagina met zoekresultaten en de snelheidseisen die we hadden opgesteld voor het tonen van een lijst. De tagcloud op de pagina met zoekresultaten is nu geïmplementeerd op basis van een vaste lijst, Mondosoft is hiervoor een API aan het ontwikkelen die pas in de volgende versie van Ontolica beschikbaar is. De snelheidsproblemen kunnen opgelost worden door caching te gebruiken.

Het prototype kan verder worden ontwikkeld tot een volwassen product. Extra functionaliteiten die gewenst zijn: rating kunnen wijzigen, tags kunnen verwijderen en tagcloud op siteniveau. Een rating veranderen kan technisch gezien al, maar mist de functionaliteit in de GUI. Het aanbieden van een tagcloud op siteniveau en de caching van tagclouds kan als één onderdeel worden geïmplementeerd.

# 8 Aanbevelingen

Bij het ontwikkelingen van nieuwe componenten in SharePoint is het noodzakelijk dat er gebruik wordt gemaakt van prototyping. Dit is nodig om te bekijken of een ontwikkeling mogelijk is, het is niet afdoende om enkel naar de API of documentatie te kijken omdat deze niet altijd correct werkt.

Het installeren en configureren van een standaard laptop met alle software voor een ontwikkelaar kost ongeveer een halve dag. Dit zou gereduceerd kunnen worden als er naast de standaard configuratie van een laptop ook een aparte configuratie voor ontwikkelaars zou worden gemaakt. Er wordt soms gedacht dat een dergelijke configuratie pas rendabel is met een groot aantal gebruikers, maar wij zijn van mening dat zelfs met een klein aantal medewerkers het voordeel direct merkbaar is. Het is dan ook makkelijker om hardware te upgraden en deze meer door te schuiven in de organisatie. Daarnaast wordt de besparing groter bij de huidige continue groei van nieuwe ontwikkelaars bij Tam Tam.

Het is interessant om kant-en-klare voorbeelden te hebben van verschillende SharePoint-technieken. Verschillende beschikbare technieken worden niet gebruikt omdat het veel tijd kost om uit te zoeken hoe ze werken. Zo wordt de veiligheid van een portal standaard op 'Volledig vertrouwen' gezet in plaats van de standaardinstelling 'Gelimiteerd vertrouwen', omdat er geen security policies voor solutions gebruikt worden. Voor een ontwikkelomgeving is dit nog te verantwoorden, maar in een productie-omgeving is dit onverstandig. Het uitwerken van technieken in direct (her)bruikbare voorbeelden zou de het gebruik van deze technieken kunnen bevorderen.

# 9 Evaluatie

In dit deel van het document geven we een korte evaluatie op de projectfasen, de documentatie, de taakverdeling en de tussentijdse rapportage.

## 9.1 Opdrachtomschrijving

Na een brainstormsessie bij Tam Tam voor dit project hebben we zelf de opdrachtomschrijving geschreven. In de eerste versie daarvan was door ons niet duidelijk beschreven hoe we het systeem dachten te gaan ontwikkelen. Zelf waren we van plan een systeem buiten SharePoint te maken en voor integratie met SharePoint te zorgen. De begeleiders bij Tam Tam hadden uit ons document echter begrepen dat we het systeem in SharePoint zouden bouwen. Dit verschil werd pas duidelijk bij de eerste keer dat we onze begeleiders hierover hebben gesproken. Wanneer we onze oorspronkelijke plannen duidelijker hadden vastgelegd, hadden we aan het begin van het project tijd kunnen besparen.

## 9.2 Ontwerp

Omdat we niet bekend waren met SharePoint en de mogelijkheden hiervan hebben we vooral in het begin van het project veel research gedaan. Om onderdelen te testen zijn er prototypen gemaakt. Bij het maken van de prototypen werd niet altijd enkel een klein onderdeel getest, vaak maakte we ook een uitbreiding van de bestaande prototypen. Het was beter geweest om de prototypen los te maken en er enkel kleine stukken functionaliteit in te testen. Het integreren van prototypen heeft wel als voordeel dat er een aantal keren problemen aan het licht zijn gebracht.

Deze uitgebreide prototyping heeft het effect gehad dat delen van het ontwerp in de prototypes ontstaan zijn in plaats van in het ontwerptraject. Het ontwerp is hierdoor op sommige punten een ontwerp achteraf geweest.

Voor de interne dataopslag is een eigen formaat gekozen. Achteraf gezien hadden we er ook voor kunnen kiezen de data als XML op te slaan, waar we minder eigen code voor hadden hoeven te schrijven. Bovendien zou een XML-formaat makkelijker uit te breiden en te exporteren zijn. Daartegenover staat dat het formaat meer opslagruimte inneemt, alhoewel de extra benodigde opslagruimte niet significant zal zijn ten opzichte van de totale interne SharePoint-database. Bij het ontwerpen is echter niet in ons opgekomen om XML te gebruiken. Dit had ons tijd kunnen besparen en was voor het maken van een prototype waarschijnlijk een betere keuze geweest.

## 9.3 Implementatie

We geven eerst een evaluatie over de manier waarop we in JavaScript hebben geprogrammeerd, waarna we een evaluatie geven van de ontwikkeling op het SharePoint-platform.

### 9.3.1 JavaScript

Voor een actieve userinterface is JavaScript nodig. We dachten deze taal goed te beheersen, omdat we hiervoor al aardig wat JavaScript geschreven hadden. Dit idee bleek echter niet te kloppen. Pas na de statische analyse van de JavaScript-code (gedaan om syntaxfouten te vinden) zagen we dat we verkeerde aannames over de taal hadden gemaakt. JavaScript is een klasse-vrije object-geörienteerde taal[6], maar met genoeg kennis zijn veel concepten van klassen uit het object-geörienteerd programmeren wel te gebruiken in JavaScript[7]. Dit hebben we onvoldoende gedaan. We hadden achteraf gezien meer object-geörienteerd kunnen werken.

### 9.3.2 SharePoint

Microsoft Office SharePoint Server 2007 is een redelijk nieuw product. Dit heeft als nadeel dat er weinig informatie over is te vinden buiten de documentatie van Microsoft om. Omdat versie 2007 op bepaalde punten sterk afwijkt van de vorige versie, zijn veel oude voorbeelden niet meer bruikbaar. Dit vormt vooral een probleem bij specialistische onderwerpen, het aantal developers wat hierover schrijft is zeer klein en er zijn meestal meer vragen dan antwoorden. Daarnaast is de documentatie van Microsoft geregeld inconsequent of leeg[8,9] en lijken verschillende API-onderdelen niet correct te werken.

## 9.4 Tests

In ons testrapport hebben we opgemerkt dat unit tests het beste door een ander persoon zouden kunnen worden geschreven dan degene die de code heeft geschreven. Hier hebben we ons helaas niet voor alle tests aan kunnen houden. Wanneer hier van tevoren in de planning rekening mee was gehouden, zou het wel hebben gekund.

We hadden gepland om gebruikersvriendelijkheidtests en snelheidstests uit te voeren na de implementatie van het prototype, aan het eind van het project. Dit bleek achteraf een onverstandige keuze, omdat we hierna te weinig tijd hadden ingepland om ons prototype te verbeteren.

Voor de gebruikersvriendelijkheid hadden we gebruikers vroeger kunnen betrekken bij het ontwerp van de interface. Daarnaast hebben we geen gebruik gemaakt van de kennis van de Interaction Designers die bij Tam Tam werken, maar hebben we de interface naar eigen inzicht ontworpen.

[6] http://www.crockford.com/javascript/inheritance.html
[7] http://www.crockford.com/javascript/prototypal.html
[8] SPField.NoCrawl, http://msdn2.microsoft.com/en-us/library/ms466351.aspx
[9] ITransformableFilterValues Properties, http://msdn2.microsoft.com/en-us/library/ms469261.aspx

## 9.5 Documentatie

Voor ons project waren we aan het begin van plan volgens de Agile Development-methode te werken. Dit leek ons de beste keuze omdat we het SharePoint-platform niet goed kenden en in een vroeg stadium van het project geen uitspraken konden doen over het systeemontwerp. De verschillende documenten die de TU Delft verwacht bij een bachelorproject, zorgen ervoor dat het project al snel volgens het watervalmodel zal moeten lopen. Dit heeft er voor gezorgd dat de documenten niet goed aansloten op het onderzoek of de ontwikkeling waar we mee bezig waren op de momenten dat zo'n document moest worden ingeleverd.

Daarnaast hadden we moeite met het schrijven van het Object Design Document. Dit wilden we baseren op het boek van Software Engineering wat in ons studieprogramma op de TU Delft werd gebruikt. In dit boek ontbreekt een duidelijke indeling van een Object Design Document. Uiteindelijk is hier een indeling voor gebruikt uit een ander boek.

## 9.6 Taakverdeling

Aan het begin van het project hebben we veel samengewerkt tijdens het doen van research naar de mogelijkheden in SharePoint en het schrijven van prototypen. Omdat we allebei onbekend waren met het platform, was de één vaak bezig met het schrijven van een stuk code terwijl de ander SharePoint-documentatie las of op internet zocht naar voorbeelden.

Op het moment dat we genoeg basiskennis hadden van het SharePoint-platform zijn we de taken gaan verdelen. Benjamin heeft de code voor de ratings en alle JavaScript-code geschreven. Daarnaast heeft hij onderzoek gedaan naar de integratie van ons systeem in de pagina met zoekresultaten van Ontolica. Joost-Wim heeft de code voor de tagging en de meeste testcode geschreven, plus de code voor het kunnen aanmaken van wiki-pagina's op basis van binnenkomende e-mailberichten. Daarnaast heeft hij de SharePoint solutions en security policies geschreven. De rest van de code is samen geschreven.

## 9.7 Tussentijdse rapportage

In de laatste paar weken van het project hebben we steeds minder contact gehad met onze begeleiders. In het Plan van Aanpak hebben we geschreven dat er wekelijks overleg zou zijn met de begeleiders van Tam Tam. Dit is in die periode minder vaak gebeurd. Ondanks het feit dat dit niet voor problemen heeft gezorgd, zou het beter zijn om hier meer op te letten. Ook wanneer we geen vragen hadden voor onze begeleiders, hadden we een kort gesprek kunnen houden. Zo zouden de begeleiders in ieder geval weten waar we op dat moment precies mee bezig waren en of we nog op schema liepen.

# 10 Literatuurlijst

De volgende bronnen zijn tijdens het project gebruikt.


Bruegge, Bernd; Dutoit, Allen H. *Object-Oriented Software Engineering Website*.
http://wwwbruegge.informatik.tu-muenchen.de/OOSE/WebHome (2007-11-26)

Crockford, Douglas. “Classical Inheritance in JavaScript.”
http://www.crockford.com/javascript/inheritance.html (2007-11-26)

Crockford, Douglas. “JSLint - The JavaScript Verifier.” 2002
http://www.jslint.com/lint.html (2007-11-26)

Crockford, Douglas. “Prototypal Inheritance in JavaScript.” 2007-04-02
http://www.crockford.com/javascript/prototypal.html (2007-11-26)

Nielsen, Jakob. “Why You Only Need to Test With 5 Users.” 2000-03-19
http://www.useit.com/alertbox/20000319.html (2007-11-26)

Spolsky, Joel. “The Joel Test: 12 Steps to Better Code.” 2000-08-09
http://www.joelonsoftware.com/articles/fog0000000043.html (2007-11-26)

“Collaborative Application Markup Language Core Schemas”, *Microsoft Developer Network*
http://msdn2.microsoft.com/en-us/library/ms462365.aspx (2007-11-26)

“Design Guidelines for Developing Class Libraries”, *Microsoft Developer Network*
http://msdn2.microsoft.com/en-us/library/ms229042(VS.80).aspx (2007-11-26)

“MIME Type Detection in Internet Explorer”, *Microsoft Developer Network*
http://msdn2.microsoft.com/en-us/library/ms775147.aspx (2007-11-26)

“XML Documentation Comments (C# Programming Guide)”, *Microsoft Developer Network*
http://msdn2.microsoft.com/en-us/library/b2s063f7(VS.80).aspx (2007-11-26)


Technische Universiteit Delft
Tam Tam

# Requirements Analysis

Knowledge Network

Joost-Wim Boekesteijn
Benjamin W. Broersma
2007-10-05
V. 2.3

# Inhoudsopgave

# 1 Inleiding

Tam Tam is een full-service internetbedrijf en is gespecialiseerd in portals, business applicaties, marketingcommunicatie-sites en online network marketing. Op dit moment heeft Tam Tam ongeveer 90 medewerkers, verspreid over twee kantoren in Rijswijk en Utrecht.

In het bedrijf is bij medewerkers, zeker bij mensen die al langer bij Tam Tam werken, veel kennis aanwezig over technologieën, producten en methoden die worden ingezet om problemen van klanten op te lossen. Deze kennis is terug te vinden in e-mailberichten die naar interne mailinglijsten worden gestuurd, in artikelen die medewerkers op hun weblog plaatsen en in projectdocumentatie die via het intranet is te vinden. Binnen Tam Tam is er behoefte om deze kennis van medewerkers aan anderen beschikbaar te stellen. Voor nieuwe medewerkers is het bijvoorbeeld nuttig om te zien welke kennis er binnen het bedrijf aanwezig is en wat de expertisegebieden van bepaalde personen zijn.

Microsoft Office SharePoint Server 2007 wordt gebruikt als primair platform voor de ondersteuning van de bedrijfsprocessen. Het beschikbaar stellen van informatie wordt daarmee voor een deel geïmplementeerd. Alle documenten die op het intranet staan zijn te vinden via de zoekfunctie van SharePoint. Andere informatiebronnen, zoals weblogartikelen en e-mailberichten, zijn niet centraal doorzoekbaar. In de ideale situatie zou alle informatie vanaf één centraal punt doorzoekbaar moeten zijn.

## 1.1 Definities

**Feature (SharePoint)**

Een component met toegevoegde functionaliteit voor SharePoint. Features zijn eenvoudig te installeren in SharePoint.

**Item**

Eenheid van informatie in SharePoint. De informatie binnen SharePoint is vrijwel altijd opgebouwd uit lijsten van items. Een item kan een document zijn, maar ook een nieuwsbericht of een contactpersoon. Naast ingebouwde itemtypes zijn er ook zelf nieuwe soorten items te definiëren.

**Public Folder**

Map met e-mailberichten die door een groep mensen kan worden gelezen en waar door iedereen een e-mailbericht heen kan worden gestuurd. Vergelijkbaar met Usenet-nieuwsgroepen.

**Rating**

Een geheel getal van 1 tot en met 5[1]. Een slechte waardering komt overeen met 1 en een goede waardering komt overeen met 5. De rating geeft aanvullende informatie over het item en maakt het mogelijk hierop te ordenen. Ratings worden vaak visueel omgezet naar een aantal sterren.

[1] Het interval $[1, 5]$ in $\mathbb{N}$.

### SharePoint

Microsoft Office SharePoint Server 2007 (MOSS 2007) is web-gebaseerde portalsoftware waarmee informatie kan worden opgeslagen en opgevraagd. Deze informatie kan aan verschillende gebruikers beschikbaar worden gesteld. De opgeslagen informatie is doorzoekbaar en direct vanuit Microsoft Word en andere Office programma's te openen. Het is te gebruiken als (Enterprise) Content Management Systeem, met als belangrijk onderdeel het Document Management Systeem. SharePoint kan worden uitgebreid. SharePoint wordt bij Tam Tam intern gebruikt en ook aan klanten van Tam Tam aangeboden. SharePoint maakt gebruik van Windows Server, Internet Information Services en SQL Server.

### Solution (SharePoint)

Bundeling van Features die geïnstalleerd kunnen worden in SharePoint. Solutions kunnen eenvoudig worden geïnstalleerd, vernieuwd en verwijderd. SharePoint biedt de mogelijkheid de installatie en het beheer van Solutions volledig te automatiseren.

### Tag

Sleutelwoord of term dat als metadata aan een item wordt gekoppeld. Een item kan meerdere tags hebben. De tag geeft aanvullende informatie over het item en maakt het mogelijk om op sleutelwoord te filteren en zoeken.[2]

### Tag cloud

Weergave van alle tags die bij een groep items horen. De tags krijgen op basis van hun frequentie een tekstgrootte toegewezen. Tags met een hoge frequentie worden in een groot lettertype getoond. Hierdoor zijn items met de meest populaire tags snel te vinden.

### Weblog

Persoonlijke website waarop een gebruiker artikelen kan plaatsen. Vaak met de mogelijkheid om bezoekers van de website te laten reageren. Verschillende medewerkers van Tam Tam hebben een publiek weblog en plaatsen hier regelmatig artikelen op.

### Wiki

Een wiki is een website waarvan de inhoud door iedereen direct gewijzigd kan worden. Door middel van samenwerking wordt de website onderhouden.

[2] http://nl.wikipedia.org/wiki/Tag_(metadata)?oldid=9197707

# 2 Huidig systeem

Er wordt in de huidige situatie informatie opgeslagen in verschillende systemen. Zo zijn er Public Folders, weblogs, mailinglijsten en SharePoint. Er is gekeken hoe en waarvoor deze systemen gebruikt worden. Er is gekeken naar de hoeveelheid informatie in deze systemen en de groei hiervan. Deze gegevens over de grootte en groei zijn in begin september 2007 verzameld. Hier volgen de cijfers per onderdeel.

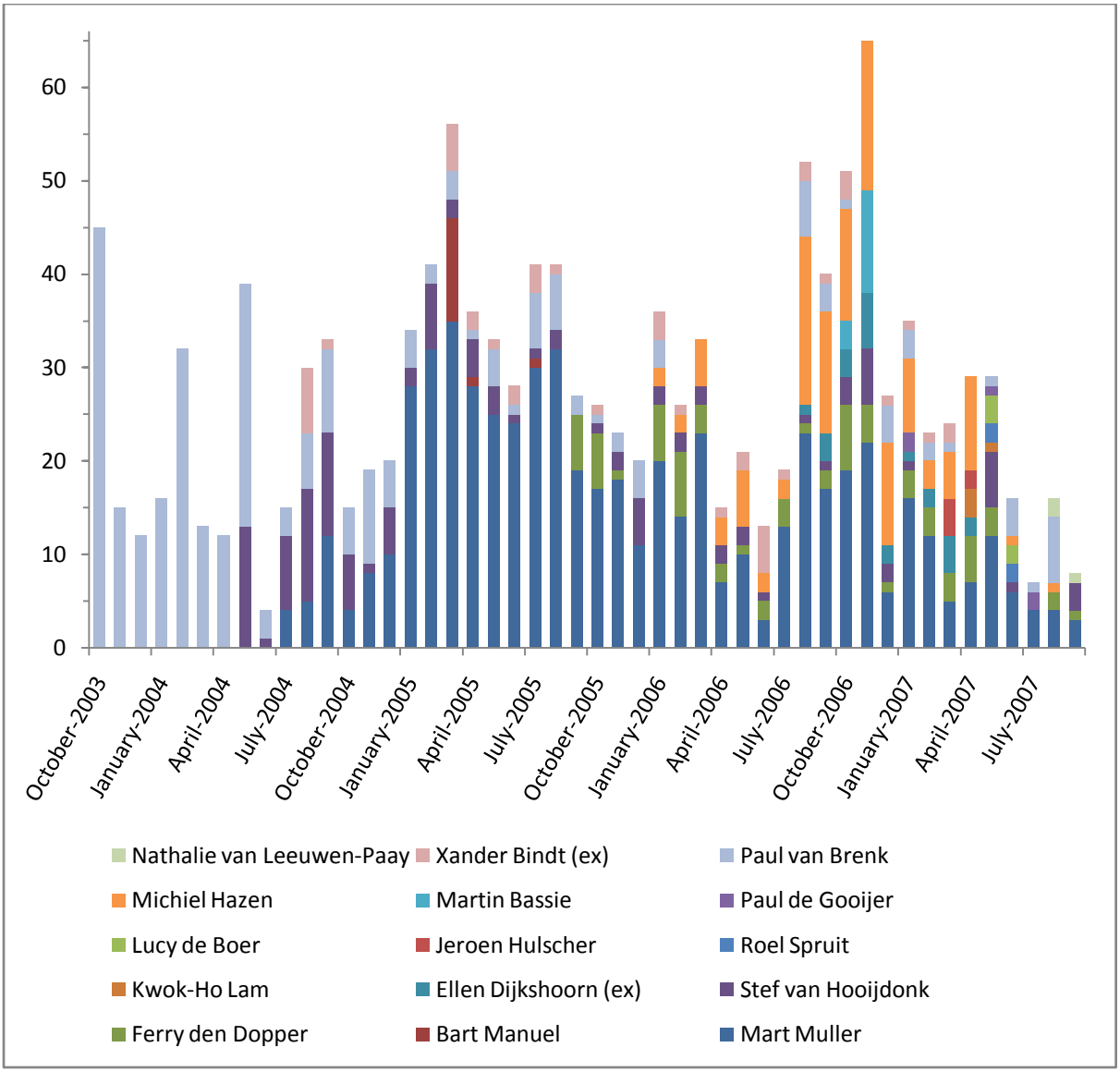
## 2.1 Public Folders

In het verleden werd veel gebruik gemaakt van Public Folders op de Microsoft Exchange Server. Na een upgrade van de Ontolica-software, die in SharePoint extra zoekfunctionaliteit biedt, kon daar echter niet meer goed in worden gezocht (vooral in meerdere mappen tegelijk) waardoor dit gebruik is verminderd. Toch worden de Public Folders nog door een enkeling gebruikt, zoals bij de servicedesk. Door één persoon worden daar de inkomende vragen en foutmeldingen gekopieerd naar een Public Folder, dit ter archivering en naslag voor de servicedesk. De overige medewerkers bij de servicedesk maken hier in de praktijk nauwelijks gebruik van. Er zijn 169 gevulde Public Folders met in totaal 7.171 berichten[3].

## 2.2 Weblogs

Door de medewerkers worden voor Tam Tam weblogs bijgehouden over verschillende onderwerpen waarmee Tam Tam zich bezighoudt. Deze weblogs zijn toegankelijk voor de buitenwereld. De weblogs worden bijgehouden in het dasBlog[4] systeem, waarvan één van de developers bij Tam Tam werkt. Op dit moment hebben 13 medewerkers van Tam Tam een weblog, samen met twee ex-medewerkers[5] hebben zij in totaal 1311 artikelen[6] geschreven. Per maand verschijnen er gemiddeld 5,5 nieuwe artikelen[7]. Eerst werden er vooral door een paar medewerkers veel artikelen geschreven. Nu worden er minder artikelen per medewerker geschreven maar zijn er wel meer medewerkers met een weblog.

[3] Telling van 5 september 2007 om 17:50, exclusief 839.387 automatisch gegenereerde systeemberichten.
[4] http://www.dasblog.info/
[5] In de grafiek aangegeven met (ex).
[6] Telling van 22 september 2007 om 12:32.
[7] Gebaseerd op gepubliceerde artikelen in de periode januari t/m augustus 2007.




**Figuur 1 Weblogartikelen per maand**

## 2.3 Mailinglijsten

Voor de kennisoverdracht worden er mailinglijsten gebruikt over diverse onderwerpen. Er zijn 80 interne mailinglijsten[8], ongeveer de helft hiervan wordt actief gebruikt. Er zijn namelijk ook mailinglijsten voor evenementen en projecten die slechts tijdelijk worden gebruikt.

Zo zijn er lijsten als “Alle Tammo’s”, “fun”, “developers”, “Information Workers”, “Online Network Marketing” en “Interaction Design”. Het aantal berichten in deze lijsten is lastig in te schatten omdat er geen centraal archief is waar deze berichten worden opgeslagen. In plaats daarvan worden deze verstuurd naar alle abonnees, welke deze vervolgens zelf beheren. Om toch een schatting te maken hebben we gekeken naar de eerste vier genoemde lijsten, waar wij op geabonneerd zijn.

| Naam | Gemiddeld aantal berichten per week |
|---|---|
| Alle Tammo’s[9] | 25,9 |
| Developers[10] | 15,4 |
| Fun[9] | 30,0 |
| SG IW[10] | 12,7 |

Tabel 1

## 2.4 SharePoint

SharePoint bestaat uit een verzameling van lijsten waar informatie in opgeslagen wordt. Deze lijsten zijn soms simpele boekenlijsten tot documentbibliotheken. Tevens zijn er wiki’s die gebruikt worden voor het beschikbaar stellen van informatie. Er zijn op het moment van tellen 7 wiki’s.

| Naam | Pagina’s[11] |
|---|---|
| Coding Convention Wiki | 66 |
| Employee Manual | 5 |
| Facility Manual | 80 |
| Microsoft kbArticles | 15 |
| Mobile device wiki | 6 |
| HTML & CSS Wiki | 3 |
| SharePoint Wiki | 25 |

Tabel 2

Daarnaast zijn er subsites voor projecten, zogenaamde Project Spaces. Een Project Space bevat altijd een documentbibliotheek met de documenten van het betreffende project; hier zijn documenten als een Functioneel en een Technisch Ontwerp te vinden. Medewerkers gebruiken bij het maken van een nieuw document vaak een goed voorbeeld van eenzelfde soort document dat eerder is gemaakt.

# 3 Te ontwikkelen systeem

Het is in het huidige SharePoint systeem niet mogelijk om vanuit de portal-site te zoeken in de weblogs, Public Folders of mailinglijsten. Het huidige systeem biedt geen mogelijkheid om aan een document een rating te koppelen, zodat documenten met een hoge rating bovenaan de zoekresultaten verschijnen. Dit is nuttig om onderscheid te maken tussen documenten die met

[8] Telling van 3 september 2007 om 11:48.
[9] Telling van 12 maart tot en met 6 september 2007 om 09:57.
[10] Telling van 24 april tot en met 6 september 2007 om 09:57.
[11] Telling van 6 september om 16:01.

dezelfde zoektermen kunnen worden gevonden, maar waarvan bepaalde documenten beter worden gewaardeerd dan de rest. Het is ook niet mogelijk om een document in een bepaalde categorie in te delen of hieraan sleutelwoorden te koppelen.

## 3.1 Verzamelen van informatie voor het project

Voor het verzamelen van de eisen van het systeem is er gebruik gemaakt van verschillende methoden. We zijn niet goed bekend zijn met de precieze mogelijkheden van het SharePoint-platform en weten daarom ook niet zeker of al onze ideeën technisch haalbaar zijn. We willen tijdens de loop van het project de mogelijkheid hebben aanpassingen te maken door nieuwe inzichten van ons of gewijzigde wensen van Tam Tam. Daarom maken we gebruik van technieken uit het Agile software development model. Er is gebruik gemaakt van brainstormsessies, interviews, gesprekken met experts en gesprekken met gebruikers. Ook is gekozen voor prototyping om snel feedback te krijgen en het project te kunnen bijsturen.

## 3.2 Oorspronkelijk plan

Vóór de start van het project hebben we met een aantal medewerkers twee brainstormsessies gehouden om ideeën op te doen voor een mogelijk project. Hierbij is nauwelijks over de technische kant gepraat, maar vooral over het concept. Aan de hand van deze brainstormsessies hebben wij de opdrachtomschrijving opgesteld, die werd goedgekeurd door Tam Tam. Na afloop hadden wij voor onszelf het idee om een los systeem te bouwen en dit te integreren met SharePoint. Bij Tam Tam was het idee om dit systeem in SharePoint te bouwen. Geen van beide ideeën was echter vastgelegd in de opdrachtomschrijving en het bestaan van deze verschillende ideeën was niet bekend.

Tijdens de eerste brainstormsessie na aanvang van het project kwamen we achter de voorkeur om het systeem in SharePoint te bouwen. We hadden echter al nagedacht over de functionaliteit van het systeem, deze functionaliteit was technisch in SharePoint niet mogelijk. Na deze brainstormsessie hebben we onderzocht in hoeverre het mogelijk was om ons oorspronkelijke idee te implementeren in SharePoint. Bij dit onderzoek hebben we gesproken met een aantal SharePoint-experts binnen Tam Tam. Met de resultaten van ons onderzoek hebben we een ontwerp gemaakt waarbij een aantal delen van het systeem los van SharePoint zouden worden gemaakt, vanwege beperkingen waar niet eenvoudig omheen te werken was. Zo kan er voor het aanpassen van de ingebouwde lijstweergave enkel gebruik worden gemaakt van de CAML View-taal, waarin geen rekenkundige bewerkingen kunnen worden uitgevoerd en de string-operaties zeer beperkt zijn. Juist omdat we hier veel aanpassingen in wilden maken, leek ons dit niet geschikt. We hadden in ons verslag ook een alternatieve oplossing gepresenteerd die was bedoeld om bij klanten van Tam Tam te implementeren waarbij we alles volledig in SharePoint zouden maken, dit product zou dan wel minder functionaliteit bevatten.

Deze nieuwe informatie hebben we gesproken met de begeleiders. De haalbaarheid van onze oplossing (het losse systeem) werd laag ingeschat en volgens hen was het risico op ’mislukking’ d.w.z. dat het niet zou worden gebruikt, erg groot. De alternatieve oplossing met minder functionaliteit, maar waarbij alles wel in SharePoint zou worden gemaakt, genoot duidelijk de voorkeur van onze begeleiders. Uiteindelijk hebben wij ervoor gekozen om deze versie te maken omdat er geen grote steun was voor het losse systeem.

## 3.3 Functionele eisen

Het te ontwikkelen systeem is een uitbreiding van SharePoint met metadata (rating, tags) waarop kan worden gezocht, gesorteerd en gefilterd. Bestaande gegevens zoals e-mailberichten in Public Folders, mailinglijsten en weblog-artikelen kunnen vanuit SharePoint worden doorzocht.

Er zijn binnen het systeem twee typen gebruikers, namelijk de gebruikers die de SharePoint portal gebruiken om informatie te zoeken, te raadplegen en toe te voegen en de beheerders die de SharePoint Solution installeren en configureren.

### 3.3.1 Gebruiker

Om gebruikers effectiever te laten zoeken naar informatie hebben we besloten metadata toe te voegen aan de informatie die aanwezig is in SharePoint. De metadata bestaat uit tags en ratings die aan elk item in SharePoint kunnen worden toegevoegd. Deze metadata wordt vervolgens gebruikt om zoekresultaten te sorteren of te filteren. Hierdoor zullen gebruikers specifiekere zoekresultaten kunnen krijgen.

De gebruiker heeft de volgende mogelijkheden:

1) Rating van een item bekijken.
2) Lijstitems sorteren op rating.
3) Rating geven aan een item.
4) Tag cloud van een lijst bekijken.
5) Tags van een item bekijken.
6) Tags toekennen aan een item.
7) Lijstitems filteren op tags.
8) Zoekresultaten sorteren op rating.
9) Zoekresultaten filteren op tags.
10) Zoekresultaten filteren op bron.

### 3.3.2 Beheerder

De beheerder heeft de volgende mogelijkheden.

1) *Installeren van de SharePoint solution*[12].
2) Beschikbaar maken van tags en ratings voor een lijst.

## 3.4 Niet-functionele eisen

### 3.4.1 Gebruiksvriendelijkheid

De interface van het systeem zal zoveel mogelijk gebruik maken van de layout en opmaak van SharePoint. De invoer van de gebruiker (het geven van tags en ratings) is van essentieel belang voor het goed functioneren van het systeem. Pas wanneer dit door gebruikers wordt gedaan, kan er beter worden gezocht in de informatie. Het is hierom belangrijk dat het voor gebruikers van SharePoint duidelijk is hoe ze tags en ratings kunnen geven en dat dit eenvoudig en snel kan gebeuren. Wanneer het bijvoorbeeld erg lang zou duren, of de interface te complex zou zijn, is het mogelijk dat gebruikers afhaken en uiteindelijk geen gebruik maken van het systeem.

[12] Dit is reeds in SharePoint aanwezig.

### 3.4.2 Documentatie

Voor ontwikkelaars die op een later moment wijzigingen maken in de code zal er documentatie worden gegenereerd uit speciaal commentaar in de geschreven code. Daarnaast zal er een installatiehandleiding komen voor SharePoint-beheerders. In het systeem zal voor de gebruiker een helpfunctionaliteit komen die het gebruik van het systeem uitlegt.

### 3.4.3 Installatie

Er zal een SharePoint Solution worden gemaakt. Hierdoor kan de installatie en het beheer op een standaard wijze gebeuren en is er slechts een kleine installatiehandleiding nodig.

### 3.4.4 Projectduur

Het Bachelorproject loopt tot en met vrijdag 12 oktober 2007. Voor deze periode zijn ongeveer 11 weken ingepland voor het project, dit komt ongeveer neer op 420 uur per persoon.

### 3.4.5 Implementatie

Voor de functionaliteit zal gebruik worden gemaakt van de beschikbare technologie in Microsoft Office SharePoint Server 2007. Dit houdt in dat er wordt gewerkt met het .NET Framework van Microsoft. Omdat SharePoint via het web toegankelijk is, zal er voor de user interface gewerkt worden met CAML, een XML-gebaseerde opmaaktaal van SharePoint, in combinatie met HTML, CSS en JavaScript. De user interface moet goed werken in Internet Explorer 6-7 en Firefox 2.

# 4 Systeemmodellen

## 4.1 Scenario's

### 4.1.1 Bob zoekt een goed functioneel ontwerp

Bob gaat een functioneel ontwerp maken voor een nieuw project en hij zoekt een goed voorbeeld. Omdat alle projectdocumentatie van vorige projecten is te vinden via de portal-site, opent hij deze site in zijn browser. In het zoekveld rechts boven op de pagina vult hij 'functioneel ontwerp' in en daarna klikt hij op de zoek-knop. Hierna verschijnt er een lijst met zoekresultaten, gesorteerd op relevantie. Deze relevantie wordt door SharePoint bepaald en hangt af van factoren zoals de frequentie van de zoekterm 'functioneel ontwerp' in de gevonden documenten en de plaats van de zoekterm in een document. Omdat het nieuwe project met SharePoint zal worden gemaakt, wil Bob enkel documenten zien die betrekking hebben op SharePoint. Hij klikt op de tag 'SharePoint' onder het kopje 'Refine by Tag'. Hierna wordt de lijst met resultaten opnieuw getoond, maar dan alleen met documenten die betrekking hebben op SharePoint. Bob ziet een aantal functionele ontwerpen staan, maar hij wil graag een goed functioneel ontwerp hebben. Naast de lijst met zoekresultaten kan hij kiezen hoe de resultaten moeten worden gesorteerd. Bob klikt op 'Order by rating' om de resultaten op hun beoordeling te sorteren. Bovenaan de lijst verschijnen een paar documenten met een goede beoordeling. Bob klikt op de titel van het eerste document, dat in Microsoft Word wordt geopend. Hij bekijkt een paar pagina's van het document. Hij is tevreden met het gevonden document en besluit dit te gebruiken als voorbeeld voor zijn eigen functioneel ontwerp.

### 4.1.2 Alice bekijkt en beoordeelt documenten

Alice is projectleider en zij verwacht dat twee collega's van haar documenten hebben opgeleverd met daarin een technisch ontwerp en een advies voor de verdere loop van het project. Omdat alle projectdocumentatie op de portal-site wordt bijgehouden, opent ze haar browser. Via haar favorieten gaat ze meteen naar de pagina met de lijst van projectdocumenten. In de lijst ziet ze twee documenten staan die gemarkeerd zijn als nieuw. Ze opent het eerste document, het technische ontwerp, door op de titel te klikken. Nadat ze het document snel heeft doorgelezen, sluit ze het en keert ze terug naar de lijst met documenten. Omdat ze wil dat het technisch ontwerp later eenvoudig is terug te vinden, besluit ze om een aantal tags toe te voegen. Ze klikt op de knop 'Tags' naast het item waarna er een scherm verschijnt waarin ze tags kan toevoegen. Ze voert de tags 'technisch ontwerp', 'SharePoint' en 'universiteit' in omdat dit een SharePoint-project voor een universiteit is. Na het invoeren van de tags klikt ze op 'Add Tag'. In de lijst met documenten verschijnen de nieuwe tags naast het document. Hierna opent ze het andere document op dezelfde manier, leest ze het door en sluit ze het weer. Ze heeft geen zin om tags toe te voegen, maar wil het tweede document wel een beoordeling geven. In de 'Rating'-kolom geeft ze een rating van drie sterren (van de vijf).

## 4.2 Use cases

In de use cases worden een aantal SharePoint-termen gebruikt die bij het lezen van de tekst wellicht niet direct duidelijk zijn. Hierom zijn in bijlage 5.1 screenshots opgenomen van een Document Library waar kolommen aan worden toegevoegd. Er zijn veel use cases die na het uitvoeren niet voor blijvende veranderingen zorgen, hiervan zijn de postcondities aangegeven met een streepje.

### 4.2.1 Ratings & tags beschikbaar stellen voor een lijst

Actoren: Beheerder

Preconditie: De beheerder heeft toegang tot een SharePoint-site.
Op de site is de Document Library 'Info' aanwezig.

Stappen:

1. **Beheerder**: opent de SharePoint-site in zijn browser.
2. **Systeem**: toont de homepage van de SharePoint-site.
3. **Beheerder**: klikt in het linkermenu onder het kopje 'Documents' op 'Info'.
4. **Systeem:** toont de lijstweergave van de Document Library 'Info'.
5. **Beheerder**: klikt op het menu 'Settings' en kiest voor 'Document Library Settings'.
6. **Beheerder**: klikt op 'Add from existing site columns' onder het kopje 'Columns'.
7. **Beheerder**: kiest uit de categorie 'Knowledge Network' de kolommen 'Rating' en 'Tags' en voegt deze toe aan de lijst 'Columns to add' en klikt hierna op 'OK'.
8. **Systeem**: voegt de kolommen 'Rating' en 'Tags' toe aan de Document Library 'Info'.

Postconditie: Gebruikers kunnen items in de Document Library 'Info' beoordelen en er tags aan toekennen.

### 4.2.2 Een lijst openen

Actoren: Gebruiker

Preconditie: De gebruiker heeft toegang tot een SharePoint-site.
Op de site is de Document Library 'Info' aanwezig met daarin enkele documenten.

Stappen:

1. **Gebruiker**: opent de SharePoint-site in zijn browser.
2. **Systeem**: toont de homepage van de SharePoint-site.
3. **Gebruiker**: klikt in het linkermenu onder het kopje 'Documents' op 'Info'.
4. **Systeem**: genereert een tag cloud op basis van alle tags van items in de lijst (wanneer de tag cloud is ingeschakeld voor deze lijstweergave).
5. **Systeem:** toont de lijstweergave van de Document Library 'Info'.
6. **Gebruiker**: ziet een lijst met documenten, met per document de huidige gemiddelde rating en de meest populaire tags (maximaal drie). Indien ingeschakeld is er ook een tag cloud te zien onderaan de lijst.

Postconditie: De lijstweergave wordt aan de gebruiker getoond.

### 4.2.3 Tags toekennen aan een item

Actoren: Gebruiker

Preconditie: De lijst 'info' is geopend (4.2.2) met daarin Item X.

Stappen:

1. **Gebruiker:** klikt naast item X op een knop om een tag toe te voegen.
2. **Systeem:** laat een scherm zien waarin tags kunnen worden ingevoerd.
3. **Gebruiker:** voert een tag in en klikt op een knop om deze toe te voegen.
4. **Systeem:** voegt de tag toe aan het item X.
5. **Gebruiker:** krijgt bevestiging dat de tag is toegevoegd en keert terug naar de lijst.

Postconditie: Er is door de gebruiker een tag toegevoegd aan item X.

### 4.2.4 Rating geven aan een item

Actoren: Gebruiker

Preconditie: De lijst 'info' is geopend (4.2.2) met daarin Item X.
Item X is nog niet beoordeeld door de gebruiker.

Stappen:

1. **Gebruiker**: geeft een rating aan item X door het aantal sterren te selecteren.
2. **Systeem**: slaat de rating op bij item X.
3. **Systeem**: rekent het nieuwe gemiddelde uit, past de weergave aan en toont deze.
4. **Gebruiker**: ziet nieuwe gemiddelde rating en gewijzigde status van het item X.

Postconditie: Item X heeft een rating gekregen van de gebruiker.

### 4.2.5 Lijstitems sorteren op rating

Actoren: Gebruiker
Preconditie: De lijst 'info' is geopend (4.2.2).
Stappen:

1. **Gebruiker**: klikt op het pijltje naast de kolomnaam 'Rating' om het menu uit te klappen.
2. **Gebruiker**: klikt op 'Ascending' of 'Descending' om de items in de lijst op- of aflopend te sorteren op beoordeling.
3. **Systeem**: sorteert de lijst op de aangegeven manier en toont de nieuwe lijstweergave.

Postconditie: -

### 4.2.6 Alle tags van een item bekijken

Actoren: Gebruiker
Preconditie: De lijst 'info' is geopend (4.2.2) met daarin Item X.
Stappen:

1. **Gebruiker:** klikt op de titel van item X.
2. **Systeem:** opent de detailweergave van item X.
3. **Gebruiker**: ziet alle tags van item X.

Postconditie: -

### 4.2.7 Lijstitems filteren op tags

Actoren: Gebruiker
Preconditie: De lijst 'info' is geopend (4.2.2).
De lijst 'info' heeft de tag cloud ingeschakeld.
Stappen:

1. **Gebruiker**: klikt op een tag in de tag cloud.
2. **Systeem**: laat alleen de items in de geopende lijst zien die de gekozen tag bevatten.

Postconditie: -

### 4.2.8 Een zoekopdracht uitvoeren in de SharePoint-site[13]

Actoren: Gebruiker
Preconditie: De gebruiker heeft toegang tot een SharePoint-site.
Stappen:

1. **Gebruiker**: opent de SharePoint-site in zijn browser.
2. **Systeem**: toont de homepage van de SharePoint-site.
3. **Gebruiker**: vult in het zoekveld rechtsboven een zoekterm in en klikt op 'Search'.
4. **Systeem**: doorzoekt alle geïndexeerde documenten op de zoekterm.
5. **Systeem**: laat de lijst met zoekresultaten zien (documenten waar de zoekterm in voorkomt). De gebruiker wordt de mogelijkheid geboden om deze lijst te sorteren en te filteren op een aantal criteria.

Postconditie: -

### 4.2.9 Zoekresultaten sorteren op rating

Actoren: Gebruiker
Preconditie: Er is een zoekopdracht uitgevoerd (4.2.8) en er worden zoekresultaten getoond.
Stappen:

1. **Gebruiker:** klikt bij de links boven de zoekresultaten op 'Sort by Rating'.
2. **Systeem:** sorteert de zoekresultaten op rating (hoogste eerst) en laat deze zien.

Postconditie: -

### 4.2.10 Zoekresultaten filteren op tags

Actoren: Gebruiker
Preconditie: Er is een zoekopdracht uitgevoerd (4.2.8) en er worden zoekresultaten getoond.
Stappen:

1. **Gebruiker**: kiest uit de lijst onder het kopje 'Refine by Tag' een tag en klikt hierop.
2. **Systeem**: laat zoekresultaten zien die de gekozen tag bevatten.

Postconditie: -

### 4.2.11 Zoekresultaten filteren op bron

Actoren: Gebruiker
Preconditie: Er is een zoekopdracht uitgevoerd (4.2.8) en er worden zoekresultaten getoond.
Stappen:

1. **Gebruiker**: kiest uit de lijst onder het kopje 'Refine by Source' een bron (Weblog, E-mail) en klikt hierop.
2. **Systeem**: laat de zoekresultaten zien die uit de gekozen bron afkomstig zijn.

Postconditie: -

[13] Deze zoekfunctionaliteit wordt reeds geboden door SharePoint en Ontolica.

## 4.3 Objectmodel

### 4.3.1 Data dictionary

Naast de standaard termen van het data dictionary is er gekeken naar de relaties tussen de termen en de acties die kunnen worden uitgevoerd.

#### *4.3.1.1 Termen*

**User**

Gebruiker van een SharePoint-omgeving, geïdentificeerd door een uniek nummer.

**ListItem**

Element in een lijst. Afhankelijk van het lijsttype kan elk ListItem een aantal kolommen met data bevatten.

**Rating**

Een geheel getal van 1 tot en met 5 (zie 1.1 Definities).

**Tag**

Sleutelwoord of term (zie 1.1 Definities).

#### *4.3.1.2 Relaties*

**ListItem heeft Ratings**

Alle Ratings die door Users zijn gegeven aan een ListItem.

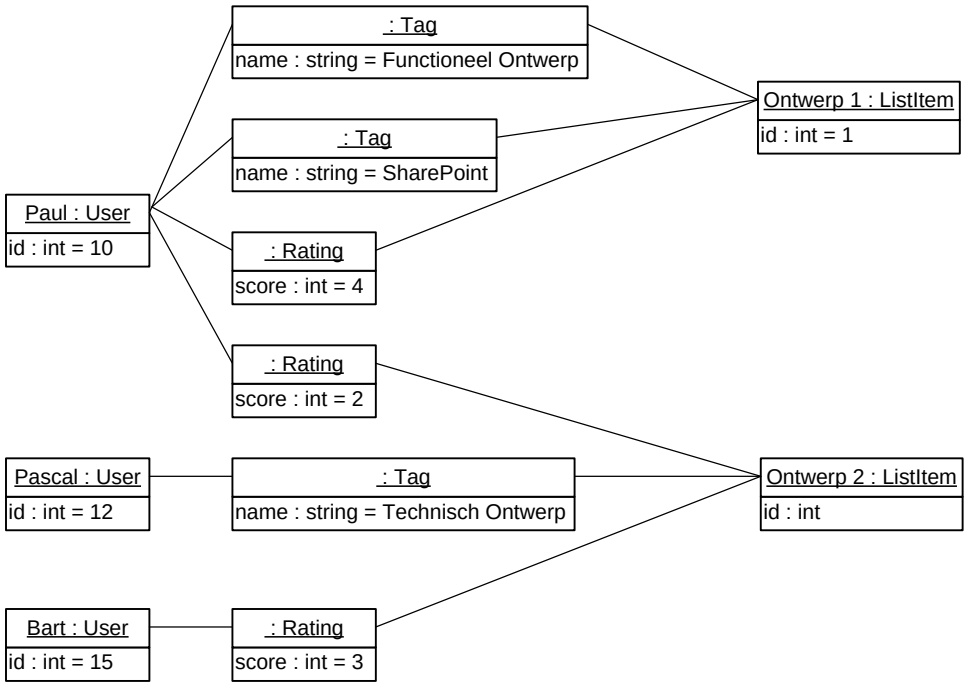
*Invariant*: een User kan een ListItem slechts één Rating geven.

**ListItem heeft Tags**

Alle Tags die door Users zijn toegevoegd aan een ListItem.

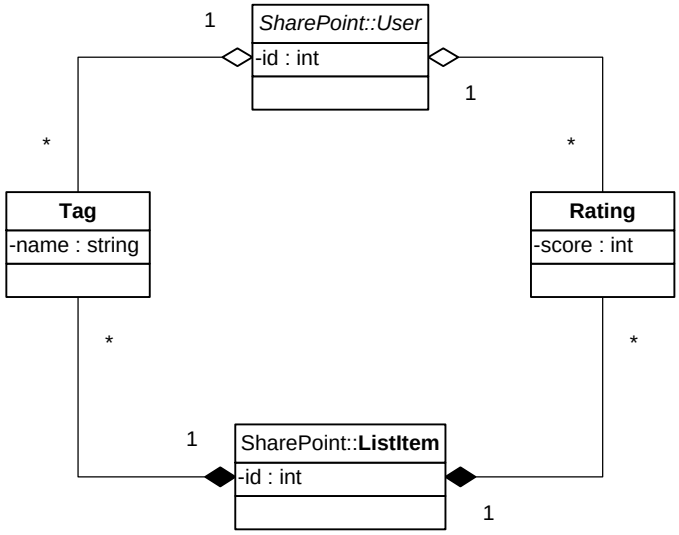
*Invariant*: een User kan meerdere tags aan een ListItem toevoegen.

### 4.3.2 Objectdiagram




UML Diagram 1

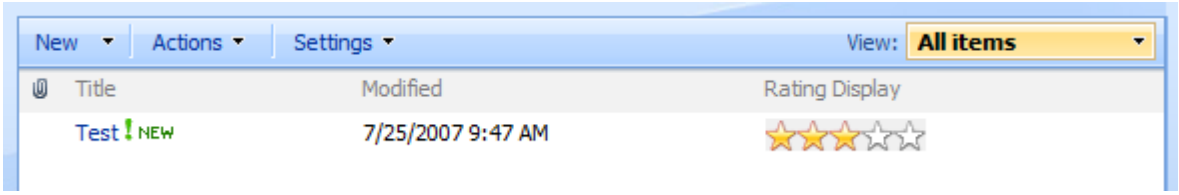
### 4.3.3 Klassendiagram



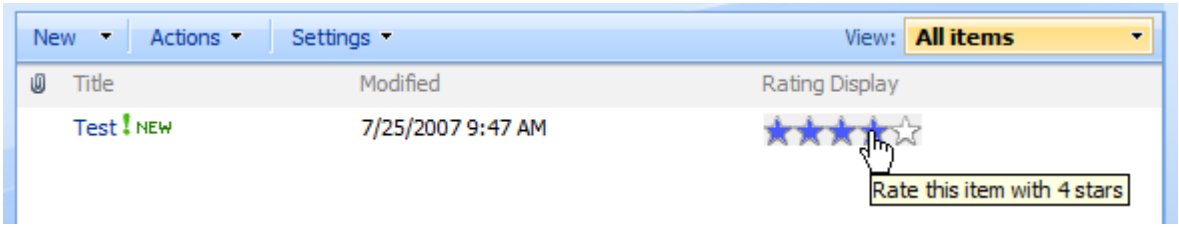

UML Diagram 2

## 4.4 Gebruikersinterface

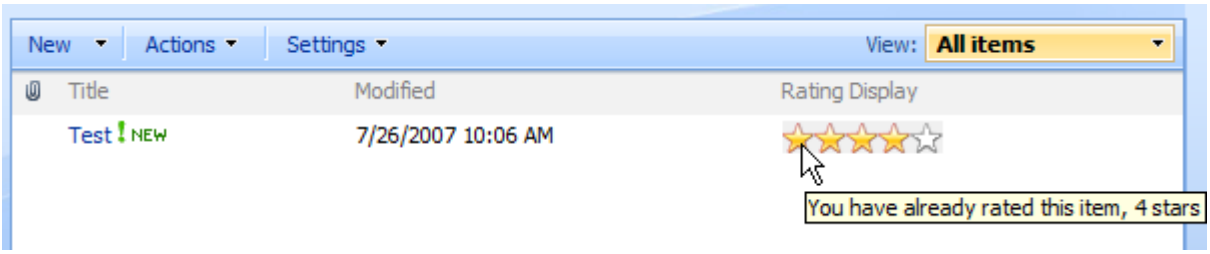
### 4.4.1 Ratings




De weergave van een item met rating in de standaard lijstweergave van SharePoint. Dit item is al beoordeeld en heeft drie sterren als gemiddelde beoordeling.

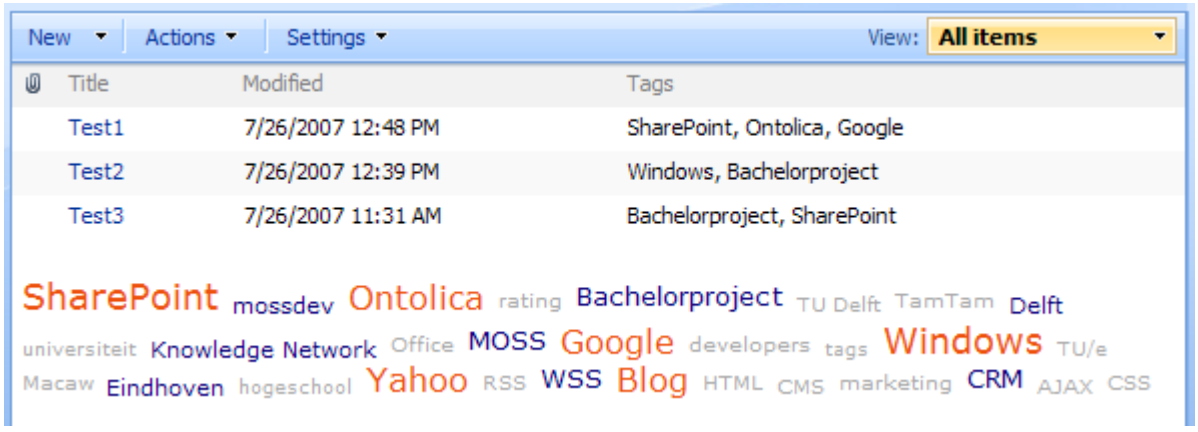



Wanneer de gebruiker met de muis over de sterren beweegt om een beoordeling te geven, licht het aantal sterren op dat als beoordeling zal worden gegeven.




Na het geven van een beoordeling wordt dit aangegeven in de hulptekst zodat de gebruiker kan zien hoeveel sterren hij aan dit item heeft gegeven.

### 4.4.2 Tags

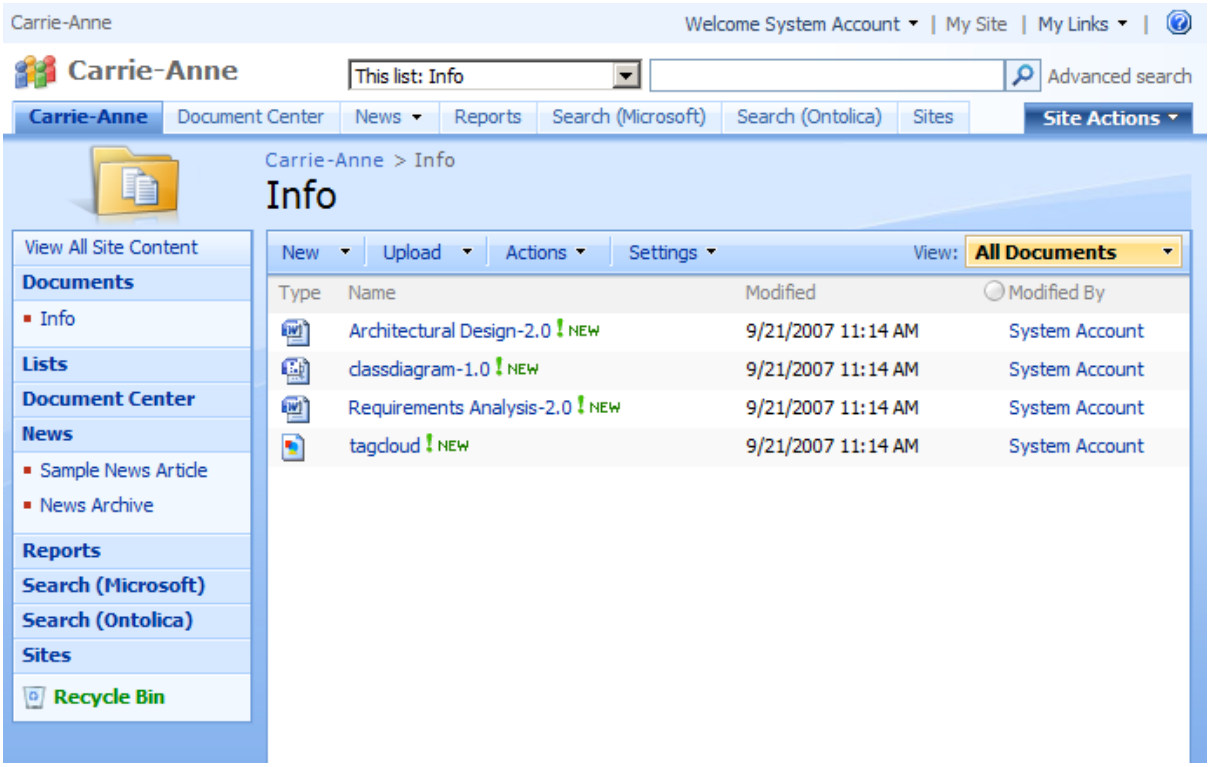



Onderaan een lijst is een tag cloud te zien, een weergave van de set van tags die zijn gegeven aan alle items in de lijst. Per item zijn de meest populaire tags te zien, maximaal drie.

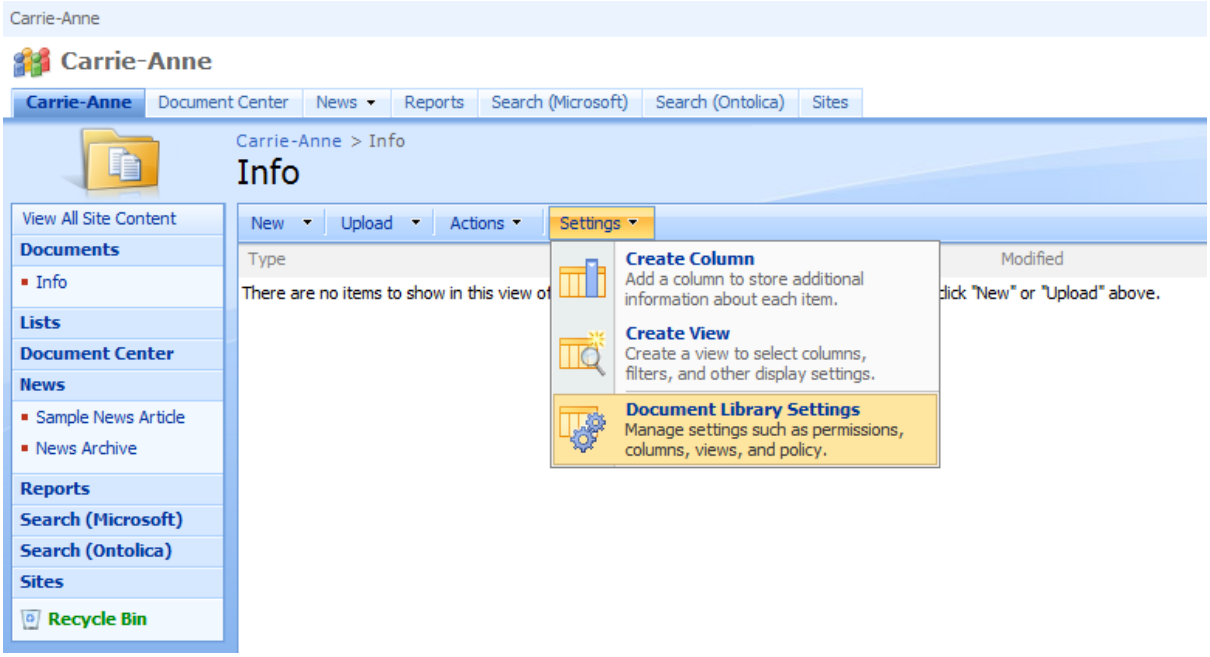
# 5 Bijlagen

## 5.1 SharePoint-screenshots

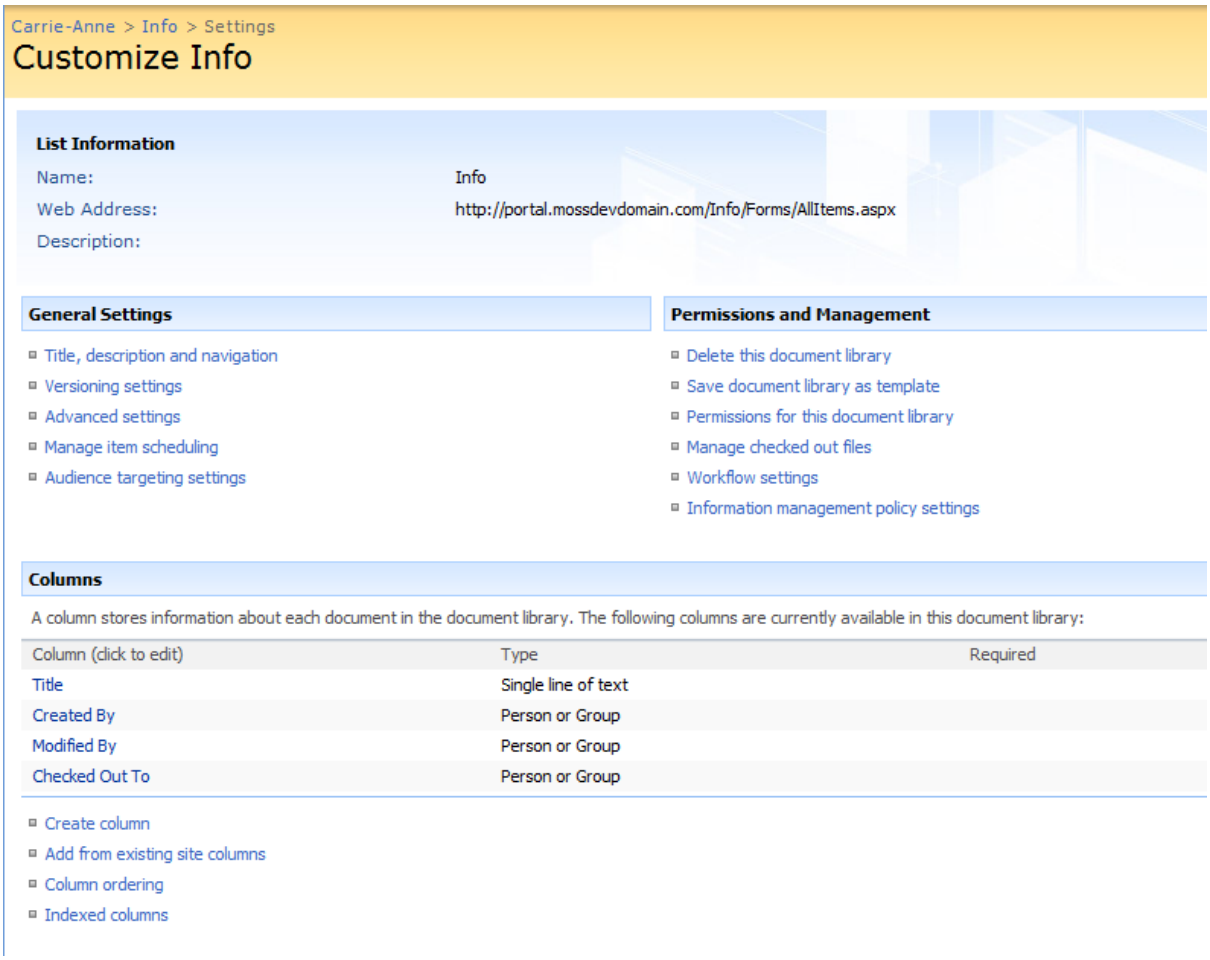
### 5.1.1 Lijstweergave van een Document Library



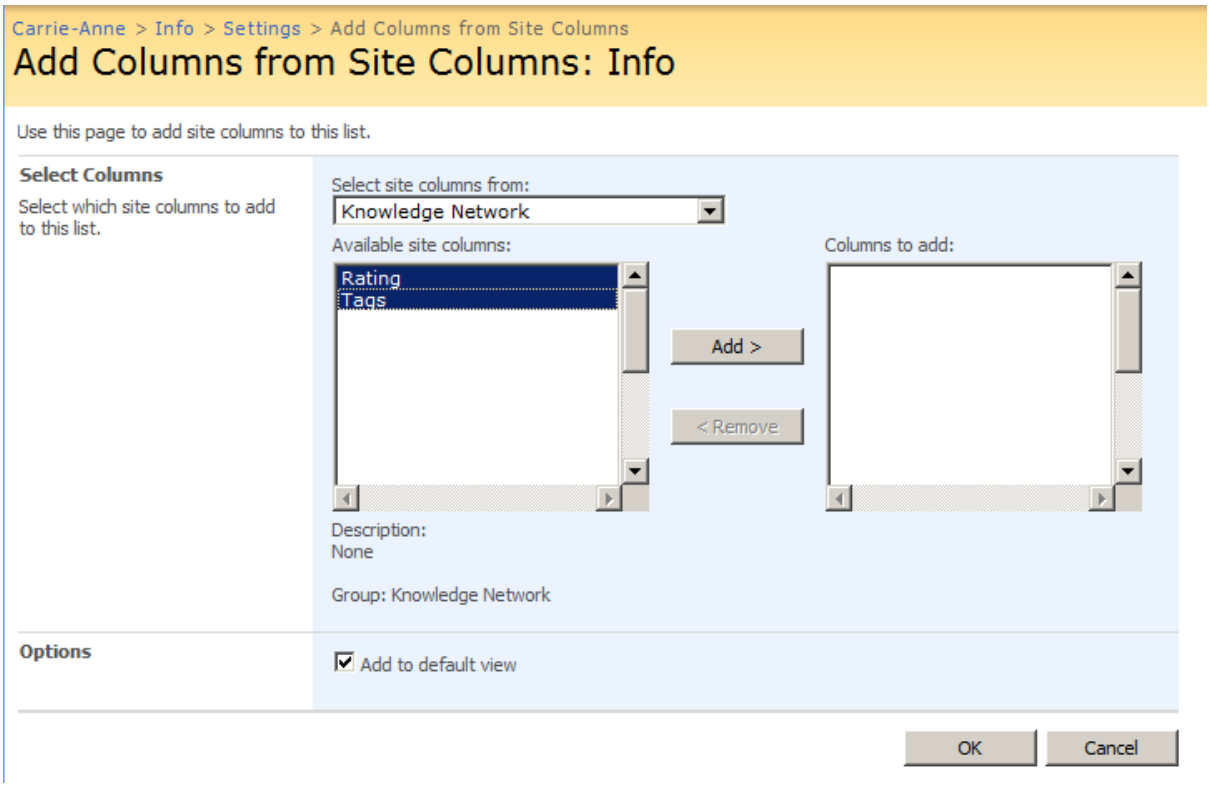

### 5.1.2 Settings-menu van een Document Library



### 5.1.3 Customize-scherm van een Document Library (voor de beheerder)




### 5.1.4 Het toevoegen van kolommen aan een Document Library



Technische Universiteit Delft
Tam Tam

# Architectural Design

Knowledge Network

Joost-Wim Boekesteijn
Benjamin W. Broersma
2007-11-12
V. 2.4

# Inhoudsopgave

# 1 Doel van het systeem

Dit document beschrijft het ontwerp van het systeem waarmee aan SharePoint tag- en rating-functionaliteiten worden toegevoegd. Om het zoeken te verbeteren, kan elke gebruiker alle data die in SharePoint is opgeslagen een waardering en een aantal tags geven. SharePoint heeft al ingebouwde zoekfunctionaliteit, deze wordt uitgebreid zodat het mogelijk wordt om bij het zoeken te kunnen sorteren op rating en te kunnen filteren op tags.

# 2 Definities

**AJAX**

*Asynchronous Javascript And XML* is een implementatietechnologie van interactieve webpagina's waarin asynchroon gevraagde gegevens worden opgehaald van de webserver. Daardoor hoeven dergelijke pagina's niet in hun geheel ververst te worden.[1]

**ASP.NET**

ASP.NET is een webapplicatie framework om dynamische websites te bouwen. Het is een onderdeel van het .NET Framework en is de opvolger van Active Server Pages (ASP). ASP.NET pagina's werken onder Internet Information Services[2].

**CAML**

Collaborative Application Markup Language[3], een XML-gebaseerde opmaaktaal van Microsoft Office SharePoint Server 2007. Er zijn twee delen, een Query- en een View-gedeelte. Het gebruikte View-gedeelte dient enkel voor het formatteren van data. Er kunnen variabelen gebruikt worden maar er kunnen geen rekenkundige bewerkingen worden uitgevoerd en de string-operaties zijn zeer beperkt. Het Query-gedeelte wordt gebruikt om data te selecteren, filteren en sorteren. Dit is een SQL query alternatief voor de SharePoint lijsten.

[1] http://nl.wikipedia.org/wiki/Asynchronous_JavaScript_and_XML?oldid=9135914
[2] http://en.wikipedia.org/wiki/ASP.NET?oldid=155839468
[3] http://msdn2.microsoft.com/en-us/library/ms462365.aspx

# 3 Ontwerpcriteria

Beslissingen in het ontwerp zijn op basis van de volgende criteria gemaakt (belangrijkste eerst): stabiliteit, veiligheid, gebruiksvriendelijkheid, schaalbaarheid en snelheid. Het prototype moet zo veel mogelijk voldoen aan deze eisen.

## 3.1 Stabiliteit

Een SharePoint pagina kan onbruikbaar worden als een component dat deel uitmaakt van een pagina een ongeldige bewerking uitvoert. SharePoint geeft de mogelijkheid componenten van een pagina uit te schakelen. Als een pagina niet werkt zullen veel gebruikers echter niet weten hoe ze dit kunnen oplossen. De componenten die geschreven zullen worden, moeten stabiel zijn en geen ongeldige bewerkingen uitvoeren zodat alle pagina's altijd beschikbaar zullen zijn.

## 3.2 Veiligheid

Het te ontwikkelen systeem wordt op basis van SharePoint ontwikkeld. SharePoint heeft een rechtensysteem waarin gebruikers en groepen gebruikers op lijst- of itemniveau een set van rechten krijgen. Informatie uit dit rechtensysteem moet worden gebruikt bij de ontwikkeling van het nieuwe systeem. Gebruikers mogen geen toegang krijgen tot data bij items waar ze geen rechten voor hebben. Ratings en tags mogen enkel worden gegeven aan items waar de gebruiker leestoegang voor heeft. De foutmeldingen die worden gegeven wanneer gebruikers onvoldoende rechten hebben om een operatie uit te voeren, moeten zorgvuldig worden gekozen om te voorkomen dat er te veel informatie wordt vrijgegeven.

Daarnaast moet het worden voorkomen dat kwaadwillende derde partijen commando's kunnen uitvoeren namens een ingelogde gebruiker of gegevens kunnen verkrijgen zonder zelf in te hoeven loggen op de website.

## 3.3 Gebruiksvriendelijkheid

De bediening van het systeem moet voor zichzelf spreken. Het systeem moet zonder uitleg gebruikt kunnen worden, iedereen moet automatisch begrijpen hoe de bediening werkt.

## 3.4 Snelheid en schaalbaarheid

De tags en rating mogelijkheid moeten het gebruik van een lijst niet vertragen. Een lijst van 500 items met rating en tags moet niet meer dan 5 seconden trager worden weergegeven. De schaalbaarheid van het systeem voor het weergeven van een lijst met tags moet van de orde $O(n \times m \times l)$ zijn waarbij $n$ het aantal lijstitems voorstelt, $m$ het aantal gebruikers is en $l$ het aantal gegeven tags is. Voor rating moet het van de orde $O(n \times m)$ zijn waarbij $n$ het aantal lijstitems voorstelt en $m$ het aantal ratings per lijstitem.

# 4 Vooronderzoek

Het oorspronkelijke plan in de opdrachtomschrijving was het maken van een Knowledge Network. Dit zou bestaan uit een Knowledge Base, waar gebruikers artikelen kunnen plaatsen en reacties en waarderingen kunnen geven. Deze Knowledge Base zou tevens de weblogartikelen, public folders en mailinglijsten bevatten. Daarnaast zou er een kennisnetwerk van medewerkers worden opgebouwd, waarbij de medewerkers in kaart worden gebracht met hun kennisgebieden, onderlinge relaties (collega's) en projectervaring.

Vanwege de voorkeur die geuit werd tijdens de kick-off van het Bachelorproject om zoveel mogelijk te integreren met SharePoint, hebben we onderzocht wat hiervan haalbaar is voor dit project. De paragrafen 4.1 tot en met 4.4 bevatten het vooronderzoek dat voor het oorspronkelijke Knowledge Network is uitgevoerd. De paragraaf 4.5 bevat de uitkomst na overleg met Tam Tam over dit vooronderzoek.

## 4.1 Data opslag

Naast data opslag mogelijkheden in Microsoft SQL Server 2005 is het in SharePoint mogelijk data op te slaan in lijsten. Deze lijsten zijn geïmplementeerd in Microsoft SQL Server. In deze lijsten kunnen standaard takenlijsten, kalenders, mededelingen en contactlijsten worden aangemaakt, ook is er de mogelijkheid zelf lijsten en datatypen te definiëren. Het grote voordeel van SharePoint-lijsten gebruiken is dat er geen afzonderlijke SQL database en tabellen aangemaakt moeten worden. De lijsten zijn vooral bedoeld om simpele data op te slaan, niet om relaties vast te leggen. Microsoft SQL Server heeft de voorkeur boven SharePoint-lijsten om zeer veel data op te slaan, dit omdat deze lijsten geïmplementeerd zijn in Microsoft SQL Server en hierdoor trager functioneren.

In het geval van het Knowledge Network is er vooral sprake van relationele dataopslag. Het gaat bijvoorbeeld om tags, ratings en views/clicks per gebruiker. Dit moet per gebruiker worden opgeslagen en er moeten veel intensieve operaties op de data worden gedaan. Hierom hebben we na overleg met Mart Muller (SharePoint Architect) besloten om voor de dataopslag voor Microsoft SQL Server te kiezen i.p.v. SharePoint-lijsten. Het nadeel is dat een SQL-gebaseerd systeem moeilijker automatisch is te installeren. SharePoint draait echter altijd op Microsoft SQL Server dus is Microsoft SQL Server altijd al aanwezig. Het is wel mogelijk met administratie- en deploymentscripts de installatie te automatiseren.

## 4.2 Integratie met SharePoint

De integratie met SharePoint voor tags, rating en reacties van artikelen is gewenst. Een gebruiker moet vanuit SharePoint aan een pagina tags en een rating kunnen geven en kunnen bekijken. Voor deze integratie is het niet noodzakelijk dat deze data ook in SharePoint wordt opgeslagen. Tags en ratings moeten het liefst toepasbaar moeten zijn voor alle items in SharePoint.

Om optimale integratie met SharePoint te bieden zullen we meerdere SharePoint-onderdelen ontwikkelen die kunnen worden aangezet om zo direct de informatie uit het Knowledge Network te gebruiken. Er is gekozen om deze tags en rating niet voor heel SharePoint aan te bieden vanwege de zoekproblemen die dit geeft. Verschillende metadata zijn niet voor alle documenten geschikt, dit wordt hieronder verder toegelicht.

## 4.3 SharePoint Search

SharePoint heeft standaard een full text search. Ontolica is een laag daarboven om meer zoekmogelijkheden te geven en zoekresultaten te verfijnen. Ontolica maakt echter wel gebruik van dezelfde onderliggende indexdata. De tags en ratings worden pas waardevol als hierop goed gezocht kan worden. Het is dus noodzakelijk om het zoekalgoritme aan te passen.

We hebben gekeken naar de technische mogelijkheden die SharePoint en Ontolica bieden om het zoekalgoritme aan te passen. Er zijn veel mogelijkheden om de weergave van de zoekresultaten aan te passen. Daarnaast is het mogelijk om eigen searchqueries uit te voeren i.p.v. de standaard searchquery en om aan verschillende onderdelen andere gewichten toe te kennen. Om geavanceerd

met tags en ratings om te gaan is er een ingrijpende aanpassing van het zoekalgoritme nodig. Een simpele herordening van de zoekresultaten biedt geen uitkomst omdat het ook mogelijk is dat de SharePoint search minder resultaten bevat dan een search gebaseerd op tags. Hoe meer het zoekalgoritme wordt aangepast, hoe minder handig het is de SharePoint search hierop aan te passen. Na overleg met Stef van Hooijdonk (Architect) zijn we vanwege deze reden tot de conclusie gekomen om voor de metadata search geen onderliggende indexdata van de SharePoint search te gebruiken. Een aangepaste SharePoint search zou het probleem hebben van vermenging van data met metadata (zoals tags en ratings) en data waarop dit niet toepasbaar is. Het samenvoegen van deze verschillende resultaten is niet praktisch. Metadata search is niet goed toepasbaar op niet-metadata. Als op een jaarverslag wordt gezocht, is het meestal niet gewenst dat er rekening wordt gehouden met de projectervaring van de auteur.

## 4.4 Future Development

Na overleg is het duidelijk geworden dat het gewenst is om aan alle items in SharePoint tags en een rating te kunnen geven. Het is dan belangrijk dat er een makkelijke installatie is. Het complete metadata zoekaspect zal hier een mindere rol spelen. Gebruikers willen vooral de mogelijkheid hebben om tags en ratings te gebruiken in SharePoint. Hiervoor zou in een simpele versie wel gebruik kunnen worden gemaakt van SharePoint-lijsten en eigen kolomtypen, dit maakt ook de installatie makkelijker. Daarnaast zou het zoekalgoritme hier in mindere mate worden aangepast dan bij de Knowledge Network search, mede omdat er ook minder metadata beschikbaar is. De gegeven tags en ratings hebben minder invloed op de zoekresultaten, maar worden bijvoorbeeld gebruikt voor het bekijken van een tagcloud op folder- en siteniveau en het sorteren van items op rating.

## 4.5 Uitkomst na overleg

Na overleg is duidelijk geworden dat wat eerst aangemerkt was als Future Development de grootste wens is van Tam Tam. Dit heeft de voorkeur omdat ‘alles’ (alle items) in de portal impliciet kennis is. De haalbaarheid van de Future Development wordt beter ingeschat dan het maken van een apart zoekalgoritme. Als het aparte zoekalgoritme niet volledig goed werkt, zou het niet gebruikt worden. Daarnaast zou de Future Development beter met SharePoint integreren.

De alternatieve oplossing, die eigenlijk was bedoeld voor klanten van Tam Tam, waarbij het systeem volledig in SharePoint zou worden gemaakt, genoot duidelijk de voorkeur van onze begeleiders. Wij hebben er voor gekozen om deze versimpelde versie in SharePoint te maken omdat er geen grote steun was voor het losse systeem. Het eerste gedeelte zal bestaan uit het volledig integreren van tags en rating in SharePoint voor alle SharePoint data. Er zal worden gekeken naar het zoeksysteem in SharePoint en er zal onderzocht worden of en hoe dit aangepast kan worden zodat er bijvoorbeeld een sortering op rating gedaan kan worden. Er zal niet veel/niets veranderd worden aan het basis zoekalgoritme dat nu wordt gebruikt.

# 5 Hoofd ontwerppunten

Het systeem wordt gebouwd in SharePoint en is gebaseerd op een webserver model. De subsysteem decompositie van het te ontwikkelen systeem zal gebaseerd zijn op het Model-View-Controller model.

## 5.1 Subsysteem decompositie

De server heeft Microsoft Windows 2003 Server als Operating System. Internet Information Services 6.0 is de webserver software. Microsoft Office SharePoint Server 2007 is gebaseerd op het .NET Framework en gebruikt SQL Server 2005 voor de dataopslag. Aan de client kant is er detail weggelaten en wordt de uitleg begonnen met de webbrowser. Globale lagen zoals een Operating System zijn hier weggelaten. De webbrowser bevat de HTML Rendering Engine en de JavaScript Engine. De donkere blokken zijn delen van het te bouwen systeem.

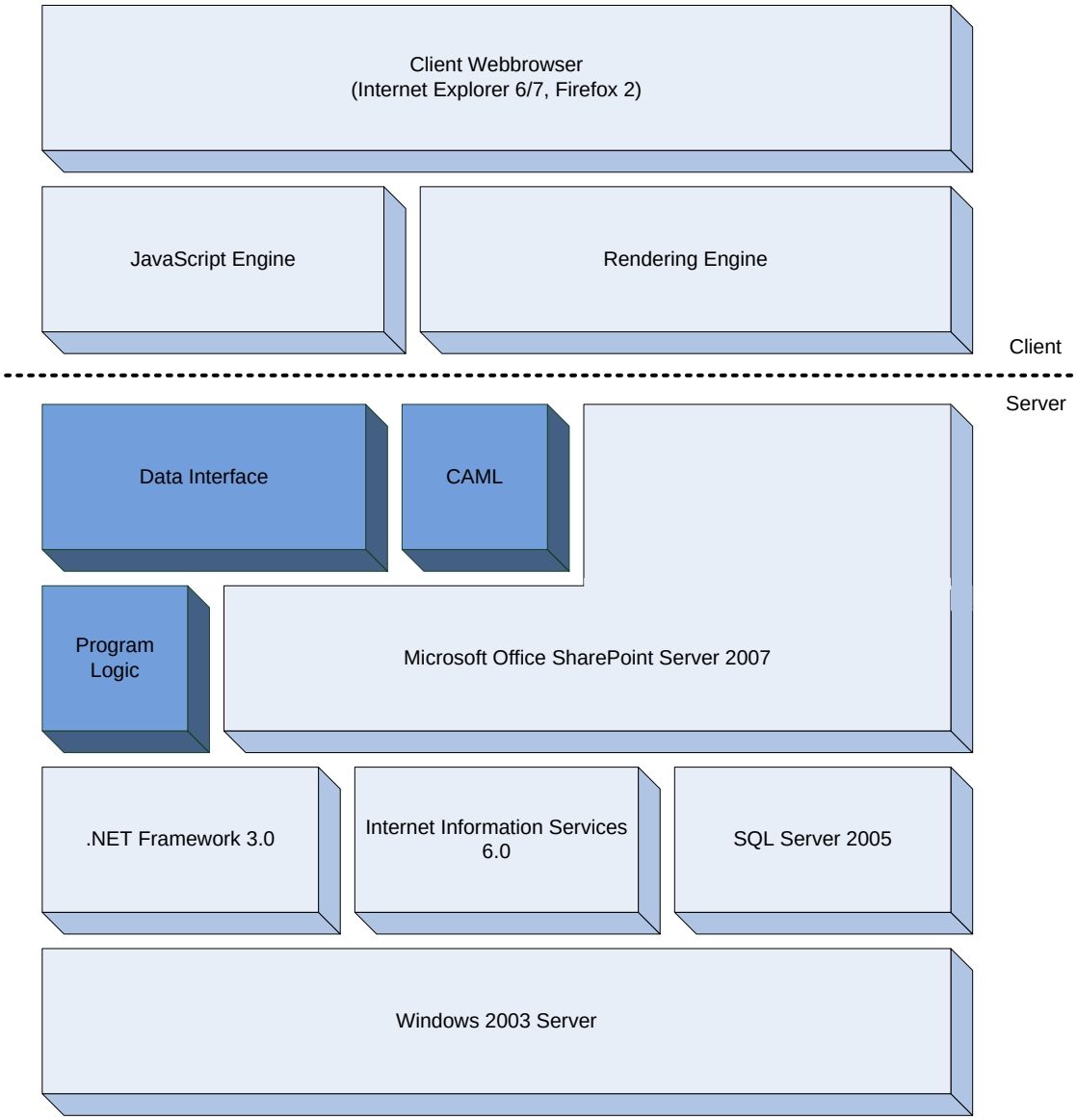



Diagram 1: Subsystem decomposition

Het schrijven van *CAML-View* code is in SharePoint nodig om een eigen weergave te maken voor een kolom, de uitvoer hiervan is HTML. Deze HTML data wordt verstuurd naar de Client Webbrowser. Als

de Client Webbrowser JavaScript ondersteunt wordt er een JavaScript-bestand geladen. Dit JavaScript-bestand voegt extra grafische effecten en *AJAX* funtionaliteit toe, zodat processen op de achtergrond kunnen worden uitgevoerd. Als er AJAX-functionaliteit beschikbaar is zal de communicatie met de Data Interface vanuit de JavaScript Engine plaatsvinden, zo niet dan zal het systeem terugvallen op directe communicatie tussen de Data Interface en de Rendering Engine.

De programmalogica heeft geen verbindingen met SharePoint en kan los gebruikt worden, deze zal geïmplementeerd worden in het .NET Framework. De programmalogica wordt aangestuurd door de Data Interface, welke geïmplementeerd zal worden in ASP.NET. Deze Data Interface is een koppeling tussen de Client Browser, de programmalogica en de SharePoint lijst opslag. Voor de dataopslag worden de ingebouwde SharePoint-lijsten gebruikt.

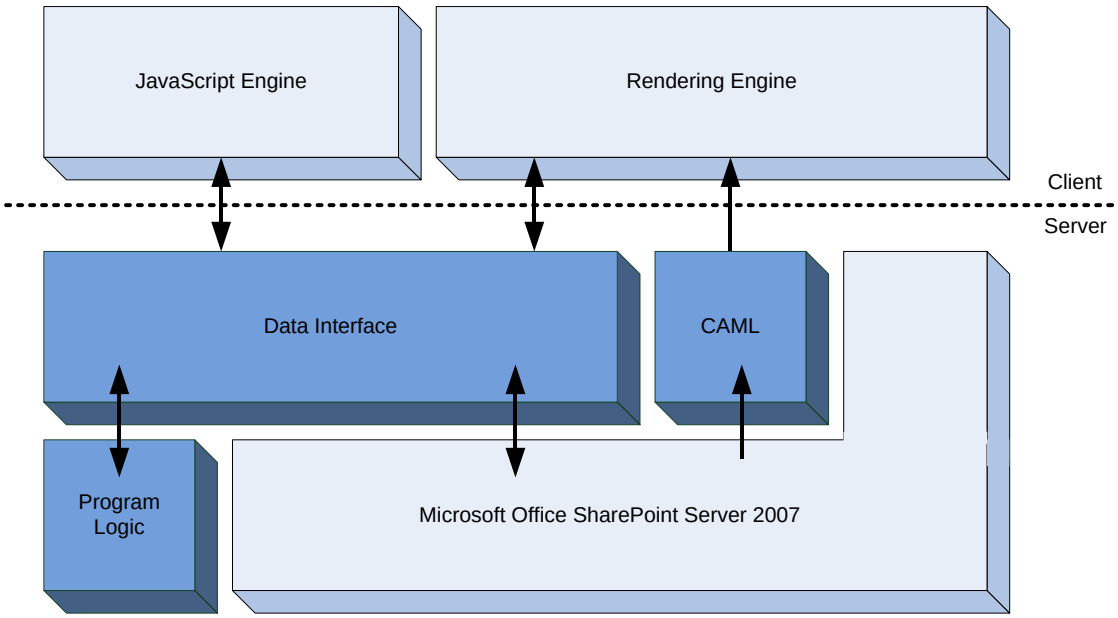



Diagram 2: Data Flow Model

## 5.2 Software mapping

Het systeem zal zo veel mogelijk geïmplementeerd worden met de beschikbare technologie in Microsoft Office SharePoint Server 2007. Dit houdt in dat er wordt gewerkt met het .NET Framework van Microsoft. Omdat SharePoint via het web toegankelijk is, zal er voor de user interface gewerkt worden met CAML, in combinatie met HTML, CSS en JavaScript. Daarnaast zal een deel van de programmalogica in ASP.NET geïmplementeerd worden. Deze verschillende delen zullen gebundeld worden in een SharePoint Feature.

## 5.3 Persistent databeheer

Naast data opslag mogelijkheden in SQL is er in SharePoint de mogelijkheid data op te slaan in lijsten. Het grote voordeel van SharePoint-lijsten gebruiken is dat er geen ‘eigen’ SQL database en tabellen aangemaakt moeten worden, wat resulteert in een eenvoudigere installatie. Er is daarom gekozen om SharePoint-lijsten te gebruiken om de tags en rating op te slaan in een kolom van de betreffende lijst.

In de lijstweergave wordt gebruik gemaakt van de CAML-View taal om een kolom weer te geven. Vanwege de beperkte mogelijkheden (zie definitie CAML) van deze taal is het noodzakelijk een

dataopslag te definiëren waar CAML de benodigde informatie uit kan halen die nodig is om de waardering van een item weer te geven. Met de tags wordt minder in de lijstweergave gedaan, enkel de meest populaire tags van een item worden weergegeven. Het dataformaat hiervoor is daarom vrijer te kiezen.

## 5.4 Globale resource handling en toegangscontrole

Een groot deel van de resource handling staat al vast door het gebruik van SharePoint. De presentatielaag van het systeem zit in de webbrowser van de gebruiker, de rest van het systeem is verspreid over verschillende server-applicaties (Internet Information Services, SQL Server, SharePoint) die op één of meerdere fysieke machines kunnen draaien. Door de scheiding van deze systemen is het voor gebruikers niet mogelijk om bijvoorbeeld de database direct te benaderen, wat veiligheidsrisico's met zich mee zou brengen.

Voor toegangscontrole en authenticatie wordt het rechtensysteem van SharePoint gebruikt. Er moet door het systeem worden gecontroleerd dat gebruikers enkel waarderingen kunnen geven en tags kunnen toevoegen aan items in lijsten waar ze leestoegang hebben. In het te ontwikkelen systeem is het een vereiste dat alle gebruikers geauthenticeerd zijn op het moment dat ze een SharePoint-pagina opvragen. Anoniem gebruik van het systeem (zonder in te loggen) wordt niet ondersteund.

## 5.5 Concurrency

In dit ontwerp speelt concurrency een rol wanneer verschillende gebruikers een lijst tegelijkertijd bewerken door een rating of tag toe te voegen of een andere lijstbewerking uit te voeren.

Als een gebruiker gelijktijdig een lijstbewerking uitvoert naast een rating of tag bewerking is er de mogelijkheid dat de lijstbewerking de oude waarde voor de rating of tag terugschrijft. Om dit tegen te gaan zal op het moment dat de itemdata wordt opgehaald het versienummer van het item worden bewaard. Dit wordt voorafgaand aan het opslaan gecontroleerd. Direct na het opslaan wordt gecontroleerd of het versienummer juist veranderd is.

Daarnaast is er het probleem dat twee gebruikers gelijktijdig een rating of tags aan een item geven en daarmee een van de rating of tags verloren gaat. Dit probleem is niet zo gemakkelijk met versienummers op te lossen. In plaats hiervan is het makkelijker om slechts één bewerking van eenzelfde item toe te staan door middel van locking/synchronisatie.

## 5.6 Grensvoorwaarden

Omdat het te ontwikkelen systeem geen zelfstandige applicatie is, maar altijd binnen de context van SharePoint wordt geladen en uitgevoerd, wordt het systeem in feite gestart en gestopt vanuit SharePoint, hier is geen controle over. Wel is er de mogelijkheid om gebruik te maken van het ASP.NET Framework welke een start en stop methode aanroept bij het starten en stoppen van de applicatie.

**Initialisatie**

Bij het starten van de communicatie naar de Data Interface (bijvoorbeeld in het geval dat een rating wordt gegeven) wordt er als er een databewerking van een item nodig is een synchronisatie[5] uitgevoerd op het te bewerken item. Vervolgens wordt de data uit SharePoint opgehaald en wordt

[5] Het locken van een item voor schrijfbewerkingen.

er indien dit nodig was een bewerking op uitgevoerd. Dit kan het toevoegen van een rating of een tag zijn.

**Afsluiten**

Indien er een synchronisatie en bewerking is uitgevoerd zal de synchronisatie worden afgerond[6] . Verder worden er geen bewerkingen meer uitgevoerd.

**Foutafhandeling**

Wanneer er tijdens het gebruik van het systeem fouten optreden, bijvoorbeeld bij het ophalen van de gegevens uit SharePoint, wordt dit weggeschreven in een logbestand en wordt er een melding naar de gebruiker gestuurd.

[6] Het vrijgeven van de lock.

Technische Universiteit Delft
Tam Tam

# Object Design

Knowledge Network

Joost-Wim Boekesteijn
Benjamin W. Broersma
2007-11-23
V. 2.2

# Inhoudsopgave

# 1 Inleiding

In het Object Design is het ontwerp van het systeem verder uitgewerkt met als uitgangspunt de indeling in subsystemen in het Architectural Design Document. Dit document volgt de opzet voor een Object Design Document zoals beschreven in 'Object-Oriented Software Engineering', Bernd Bruegge en Allen H. Dutoit, 2004.

Als eerste zullen de algemene trade-offs besproken worden die in overweging zijn genomen en invloed hebben gehad op het systeemontwerp. Daarna worden de coding conventions besproken. Vervolgens is op basis van de subsystemen uit het Architectural Design Document de vertaling naar klassen gemaakt, met bijbehorende klassendiagrammen. Van een selectie van deze klassen is een gedetailleerde specificatie opgenomen. In bijlagen zijn er implementatiedetails opgenomen over de beperkte mogelijkheden van CAML en het hieruit voortvloeiende eigen serialisatieformaat voor de opslag van de interne data.

# 2 Trade-offs

De volgende trade-offs zijn in overweging genomen en zijn gerangschikt op basis van de invloed die ze hebben gehad op het systeemontwerp.

## 2.1 SharePoint vs. Eigen systeem

Voor de start van het Bachelorproject was er bij ons de veronderstelling een eigen systeem te gaan maken met integratie met SharePoint. Bij Tam Tam was er het idee een systeem in SharePoint te bouwen. Na een onderzoek naar de mogelijkheden in SharePoint en een overleg met de begeleiders van Tam Tam hebben we de keuze gemaakt om het systeem in SharePoint te bouwen. Er is na het maken van prototypes gekozen om de data zo veel mogelijk bij de betreffende elementen (zoals een lijstitem) op te slaan. Dit heeft als voordeel dat er gebruik kan worden gemaakt van de bestaande search en geeft de mogelijkheid om deze gegevens in de lijstweergave te tonen.

## 2.2 Functionaliteit vs. Tijd

In plaats van een prototype met enkel de minimale technische functionaliteit, zal er een prototype worden gemaakt met alle gewenste functionaliteit en een user-interface die geschikt is voor eindgebruikers. Deze keuze is gemaakt zodat de opdrachtgever een duidelijk beeld kan krijgen over de technische haalbaarheid en mogelijkheden.

Het prototype is gemaakt als 'Proof of Concept'. Hierin is ontwikkeltijd belangrijker dan de snelheid van het uiteindelijke systeem. Er worden dus geen optimalisaties gemaakt voordat het prototype werkend is. Hierna kan gedacht worden aan caching van bijvoorbeeld tagclouds om de responstijd te verbeteren.

## 2.3 Kopen vs. Maken

Bij de start van het project is er kort onderzocht of er al een soortgelijk product op de markt zou zijn. Er werd enkel een simplistische rating module (met reacties) gevonden voor MOSS 2003[1]. Hierin was geen mogelijkheid tot het geven van tags, daarnaast was de rating niet met de search geïntegreerd en waren de user-interface functionaliteiten beperkt. We hebben toen toch contact opgenomen met het bedrijf om te vragen of en wanneer er een nieuwe versie zou verschijnen voor MOSS 2007. Deze

[1] SharePoint 2003 Rating & Experts Module, http://www.pilothouseconsulting.com/rating/RatingExperts.html

zou in oktober 2007 uitkomen maar is tot op heden[2] niet uitgebracht. Hierdoor hebben we besloten zelf een systeem te ontwikkelen.

# 3 Coding convention en version control

De C# coding convention binnen Tam Tam is gebaseerd op de richtlijnen zoals beschreven in de 'Design Guidelines for Developing Class Libraries', Microsoft, 2005[3]. Deze richtlijnen zijn ook in dit project gebruikt. Enkele voorbeelden hiervan:

```
/// <summary>
/// This class represents a single element in a TagCloud.
/// </summary>
public class TagCloudElement : IComparable<TagCloudElement>
{
        private string tagName;

        public string TagName
        {
                get
                {
                        return tagName;
                }
        }

        public void AddTags(Tags tags)
        {
                ...
        }
}
```

**Code snippet 1: Voorbeeldcode ter illustratie van de coding conventions**

Namen van klassen, methoden en properties[4] beginnen met een hoofdletter. Elk opeenvolgend woord in de naam begint met een hoofdletter (Pascal-case): TagCloudElement, TagName, AddTags.

Namen van member variables en argumenten beginnen met een kleine letter. Elk opeenvolgend woord in de naam begint met een hoofdletter (Camel-case): tagName, tags.

Voor namen van klassen worden zelfstandige naamwoorden gebruikt, namen van methoden beginnen met een werkwoord.

Conventies voor naamgeving van methoden of klassen uit het .NET Framework en de het SharePoint-objectmodel worden overgenomen. Hierbij moet worden opgemerkt dat de naamgeving van klassen in SharePoint niet consistent is. Soms wordt SP als prefix voor de klassenaam geplaatst, maar in een aantal gevallen juist niet. We hebben deze methode van naamgeving overgenomen.

Wanneer er in een methode een fout optreedt, wordt er een exception gegooid.

Commentaar bij de code wordt geschreven met XML Documentation Comments[5] zodat er later documentatie kan worden gegenereerd op basis van de code.

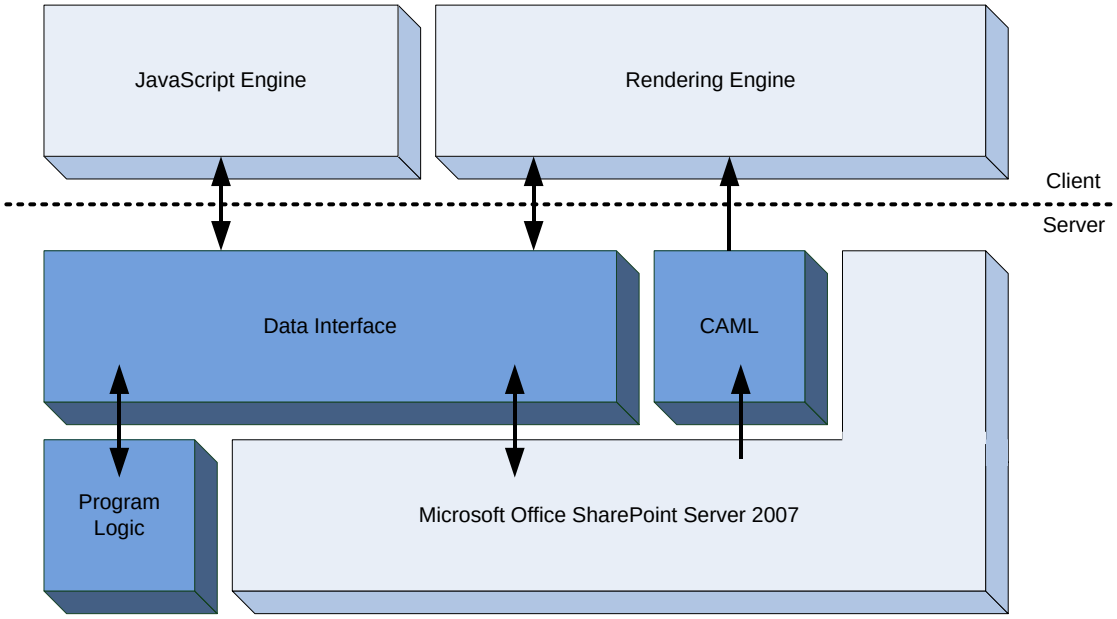
---

[2] vrijdag 23 november 2007
[3] Design Guidelines for Developing Class …, http://msdn2.microsoft.com/en-us/library/ms229042(VS.80).aspx
[4] Properties in C# zijn equivalent met getters en setters in Java.
[5] XML Documentation Comments, http://msdn2.microsoft.com/en-us/library/b2s063f7(VS.80).aspx

Voor de opslag wordt gebruik gemaakt van een version control system, hier worden alle projectbestanden (broncode, documenten) in opgeslagen. Voor broncode is er de extra eis dat er gecontroleerd moet worden of het gewijzigde project compileert zonder fouten, pas dan mogen de wijzigingen worden doorgevoerd.

# 4 Indeling van de subsystemen in klassen

In het Architectural Design Document is de volgende onderverdeling in subsystemen gemaakt:

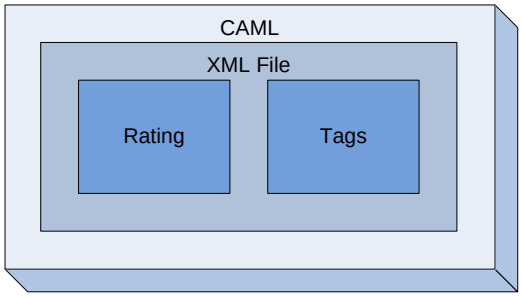



Figuur 1: Indeling in subsystemen

Per subsysteem wordt in de volgende paragrafen uitgelegd welke klassen zich in het subsysteem bevinden en welke afhankelijkheden er zijn van andere systemen. In sectie 5 worden de klassen in meer detail besproken.

## 4.1 CAML

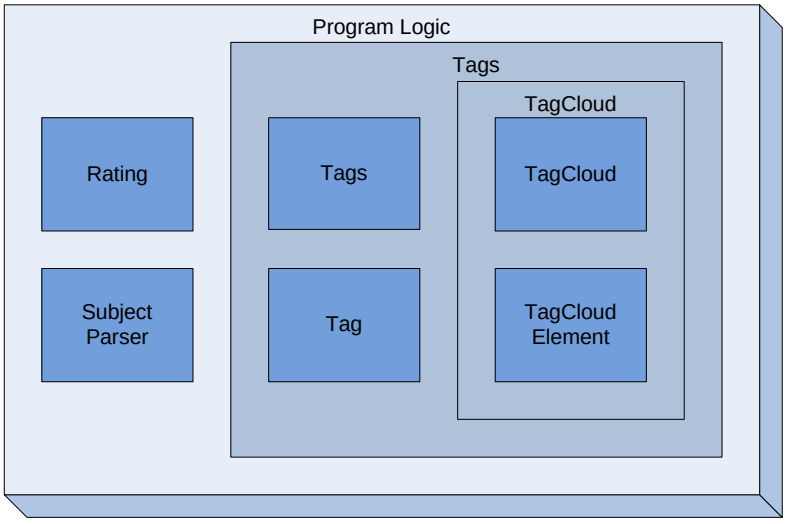
De CAML-code zorgt voor kolomsgewijze weergave van veldtypen. Deze weergave is per veldtype gedefinieerd in CAML (gebaseerd op XML). De definities voor de veldtypes Rating en Tags staan in een XML-bestand. Wanneer een SharePoint-pagina wordt geladen met een lijstweergave waarin een kolom van het type Rating of Tags voorkomt, gebruikt SharePoint de CAML-code om deze kolom te tonen. Dit is het enige moment waarop de CAML-code wordt gebruikt. In 'Bijlage B: CAML' is te zien hoe er op implementatieniveau om bepaalde beperkingen van CAML is heengewerkt.




Figuur 2: Indeling van de CAML-code

## 4.2 Program Logic

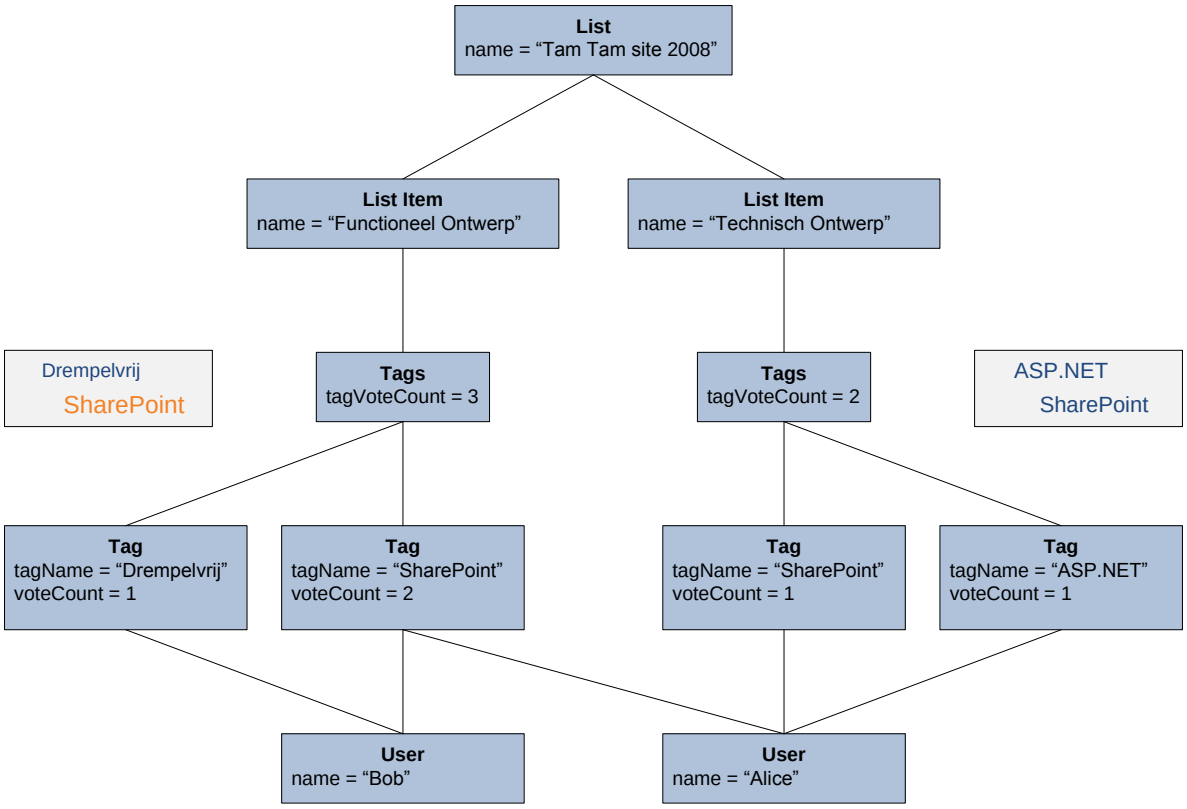
Binnen de programmalogica is er een onderverdeling te maken in de aanwezige klassen. De klassen Rating en SubjectParser staan los van de klassen die met tags werken.




Figuur 3: Klassen in de programmalogica

De klasse Rating bevat alle functionaliteit voor het kunnen geven van ratings aan items in SharePoint. De SubjectParser wordt gebruikt om informatie over rating en tags uit het onderwerp van een e-mailbericht te halen wanneer een SharePoint-lijst is ingesteld om documenten per e-mail te kunnen ontvangen.

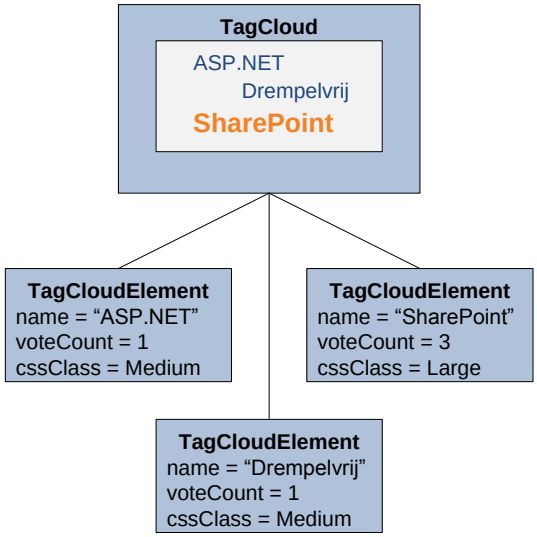
In het volgende objectdiagram zijn de relaties te zien tussen de Tags-klassen.




Figuur 4: Objectdiagram van een lijst met twee lijstitems en tagclouds op lijstitemniveau

Een lijst bevat lijstitems en een lijstitem kan meerdere tags (Tag-objecten) bevatten. In een Tag-object wordt de naam van de tag en een lijst van gebruikers opgeslagen (en daarmee het aantal stemmen). Er kan bij een lijstitem een tagcloud gemaakt worden van deze tags door alle Tag-objecten te classificeren op basis van het aantal stemmen.

Om op lijstniveau een tagcloud te maken moeten eerst alle Tag-objecten bij elkaar worden gevoegd. Tag-objecten met dezelfde naam worden dan gebundeld. Een gebruiker kan bij twee lijstitems een tag met dezelfde naam hebben toegevoegd. Hierdoor komt de lijst van gebruikers niet meer overeen met het aantal stemmen. Daarom wordt hier de TagCloudElement-klasse gebruikt, deze bevat geen lijst van gebruikers maar enkel het aantal stemmen. In de TagCloudElement klasse wordt tevens de gegeven classificering opgeslagen.




Figuur 5: Tagcloud op lijstniveau van Figuur 4

### 4.2.1 Tags

Deze klassen binnen de programmalogica zijn ontworpen voor het kunnen toevoegen van tags aan items en voor de weergave van tagclouds op verschillende niveaus.

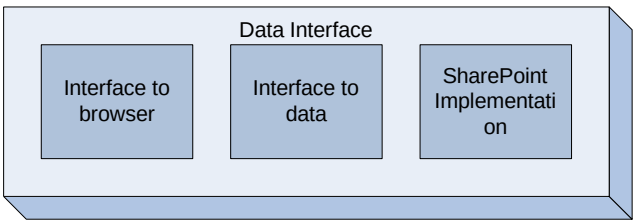
De klassen Tag en Tags bevatten samen alle informatie over een set van tags die aan een item in SharePoint zijn toegekend.

#### *4.2.1.1 TagCloud*

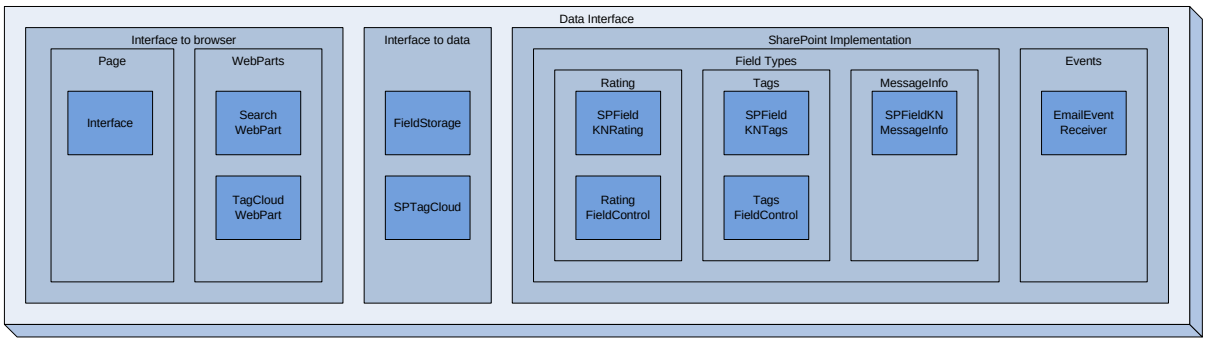
Voor het tekenen van tagclouds op lijst- en itemniveau zijn de klassen TagCloud en TagCloudElement beschikbaar. De klasse TagCloudElement bevat niet alle functionaliteit van de klasse Tag, maar heeft alleen de minimaal benodigde data en methoden om een tag in een tagcloud te kunnen tekenen. Daarnaast bevat een TagCloudElement-object extra data en methoden die enkel worden gebruikt voor het tekenen van de tag in een tagcloud.

## 4.3 Data Interface

De Data Interface laat verschillende subsystemen communiceren met de programmalogica en bevat klassen die bestaande SharePoint-klassen uitbreiden met eigen functionaliteit. Binnen de Data Interface zijn er drie groepen van klassen te onderscheiden: Interface to browser, Interface to data en SharePoint Implementation.




Figuur 6: Verschillende interfaces in de Data Interface




Figuur 7: Klassen naar interfaces in de Data Interface

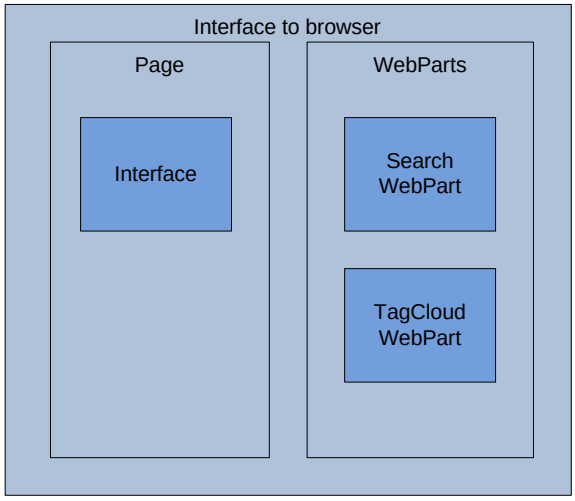
In Figuur 7 zijn alle klassen te zien in de drie groepen van klassen uit de Data Interface. In de volgende paragrafen wordt per groep uitgelegd welke functionaliteit zich in de klassen uit die groep bevindt.

### 4.3.1 Interface to browser

De interface naar de browser bevat klassen die door ASP.NET worden gebruikt voor het aanbieden van webpagina's en webparts.

#### *4.3.1.1 Page*

De klasse 'Interface' is een uitbreiding van de ASP.NET klasse Page[6]. Code in de klasse wordt uitgevoerd op het moment dat een webpagina wordt opgevraagd. Dit wordt gebruikt als een interface naar de programmalogica toe voor de webbrowser van een gebruiker. De JavaScript-code in de webbrowser communiceert met deze laag.




Figuur 8: Klassen in de interface naar de browser

#### *4.3.1.2 WebParts*

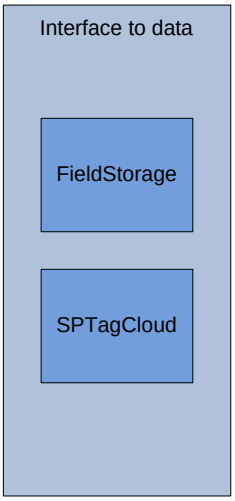
SharePoint ondersteunt het toevoegen van componenten op een pagina. Ontwikkelaars kunnen zelf componenten schrijven door een klasse te maken op basis van de WebPart-klasse uit het ASP.NET framework. Wanneer een gebruiker een pagina opvraagt, wordt er een methode aangeroepen in de WebPart-klasse. Deze methode kan uitvoer genereren door HTML te schrijven. De HTML wordt vervolgens op de pagina getoond. WebParts hebben toegang tot de SharePoint-context. Zo is het mogelijk om vanuit een WebPart gegevens op te vragen over de pagina waarop het WebPart is toegevoegd.

De webparts SearchWebPart en TagCloudWebPart laten een tagcloud zien in de weergave van de zoekresultaten en de lijstweergave. Hiervoor wordt de klasse SPTagCloud gebruikt.

[6] Page, http://msdn2.microsoft.com/en-us/library/dfbt9et1(VS.80).aspx

### 4.3.2 Interface to data

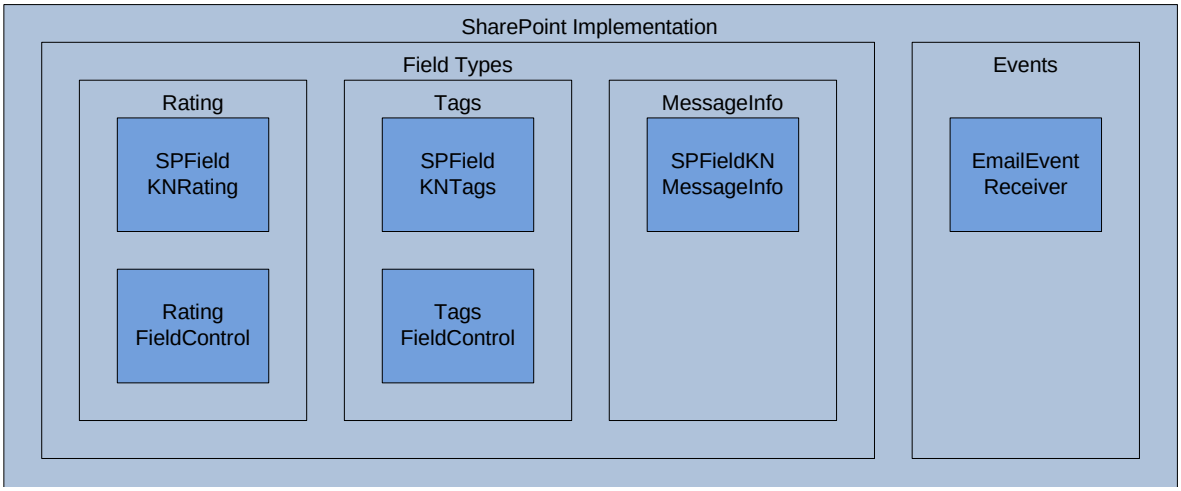
De klassen FieldStorage en SPTagCloud worden gebruikt op plaatsen waar vanuit de SharePoint-context toegang tot de programmalogica nodig is. Deze klassen bevatten functionaliteit die vanuit verschillende plekken nodig is. De SPTagCloud-klasse wordt bijvoorbeeld door de drie klassen uit de 'Interface to browser' gebruikt die alle een tagcloud produceren. De FieldStorage-klasse zorgt voor de serialisatie van de velden Tags en Rating. Het is hier nodig om met een eigen serialisatieformaat te werken zodat deze gegevens ook met CAML kunnen worden uitgelezen. Meer details over het gekozen formaat zijn te vinden in 'Bijlage A: Serialisatieformaat'.




**Figuur 9: Data-interface klassen**

### 4.3.3 SharePoint implementation

Klassen die onder de SharePoint implementatie vallen, zijn allemaal gebaseerd op ingebouwde SharePoint-klassen die kunnen worden uitgebreid om nieuwe functionaliteit toe te voegen.




**Figuur 10: Onderverdeling in klassen van de SharePoint uitbreidingen**

#### 4.3.3.1 Field Types

Voor het maken van een eigen veldtype moet een klasse worden gemaakt die is gebaseerd op de SharePoint-klasse SPField[7]. Voor het definiëren van de weergave van een kolom gebaseerd op zo'n veldtype moet een klasse worden gemaakt die is gebaseerd op de SharePoint-klasse BaseFieldControl[8]. Omdat een SharePoint-FieldControl niet in de lijstweergaven wordt gebruikt, is het nodig om daar met CAML te werken voor het tonen van kolommen met een eigen veldtype.

De veldtypen voor Rating en Tags bevatten allebei een definitie van een FieldControl, terwijl dat bij MessageInfo niet zo is. Het veld MessageInfo wordt namelijk in een kolom gebruikt die niet wordt weergegeven maar enkel voor interne opslag wordt gebruikt.

#### 4.3.3.2 Events

In SharePoint kunnen klassen worden gemaakt die bepaalde events afhandelen. Hiermee is het mogelijk om code uit te voeren wanneer er een e-mailbericht wordt ontvangen op een adres dat is gekoppeld een SharePoint-lijst. De klasse EmailEventReceiver handelt dit event af en is gebaseerd op de SharePoint-klasse SPEmailEventReceiver[9].

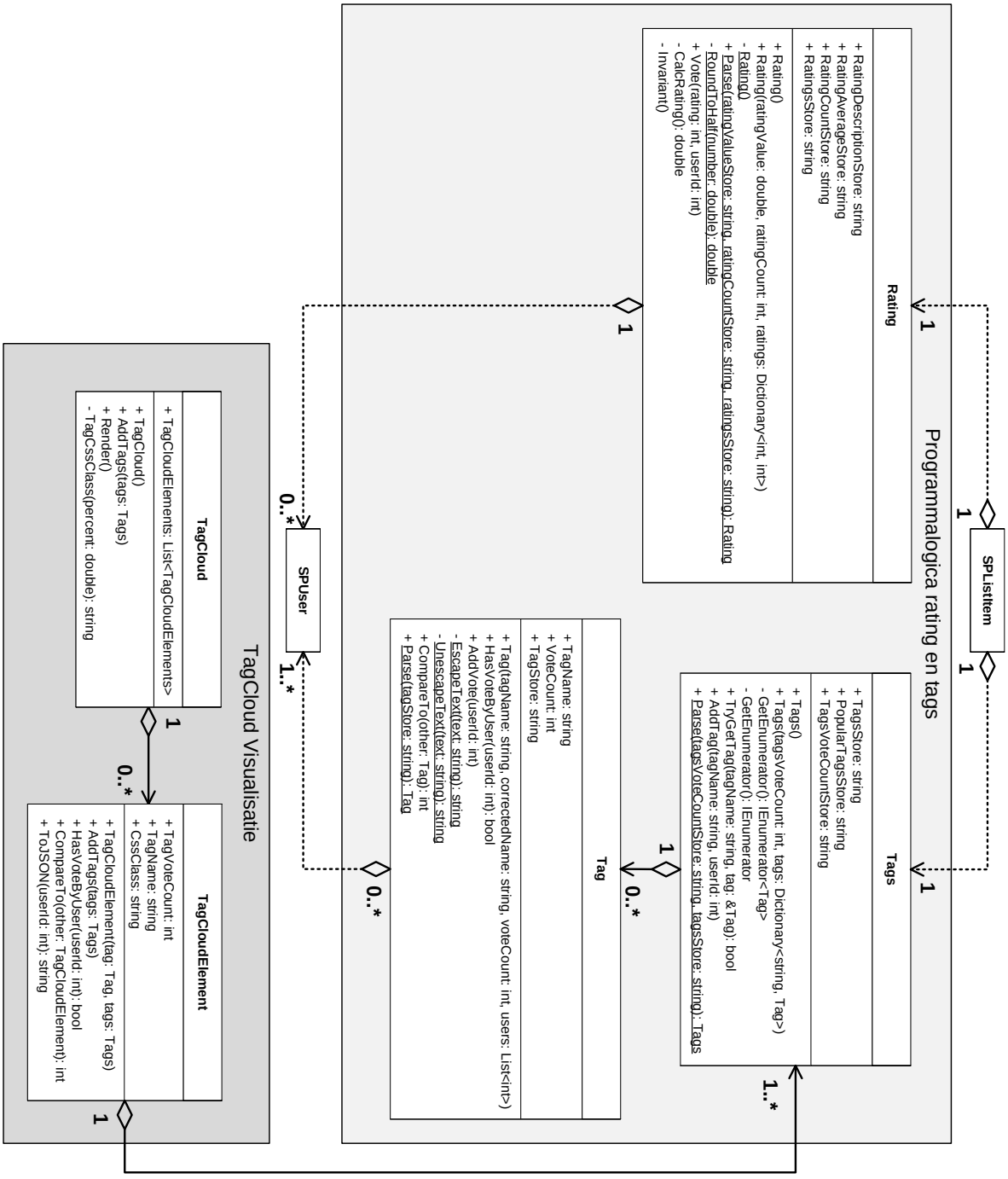
[7] SPField, http://msdn2.microsoft.com/en-us/library/ms438369.aspx
[8] BaseFieldControl, http://msdn2.microsoft.com/en-us/library/ms415539.aspx
[9] SPEmailEventReceiver, http://msdn2.microsoft.com/en-us/library/ms414575.aspx

# 5 Klassen

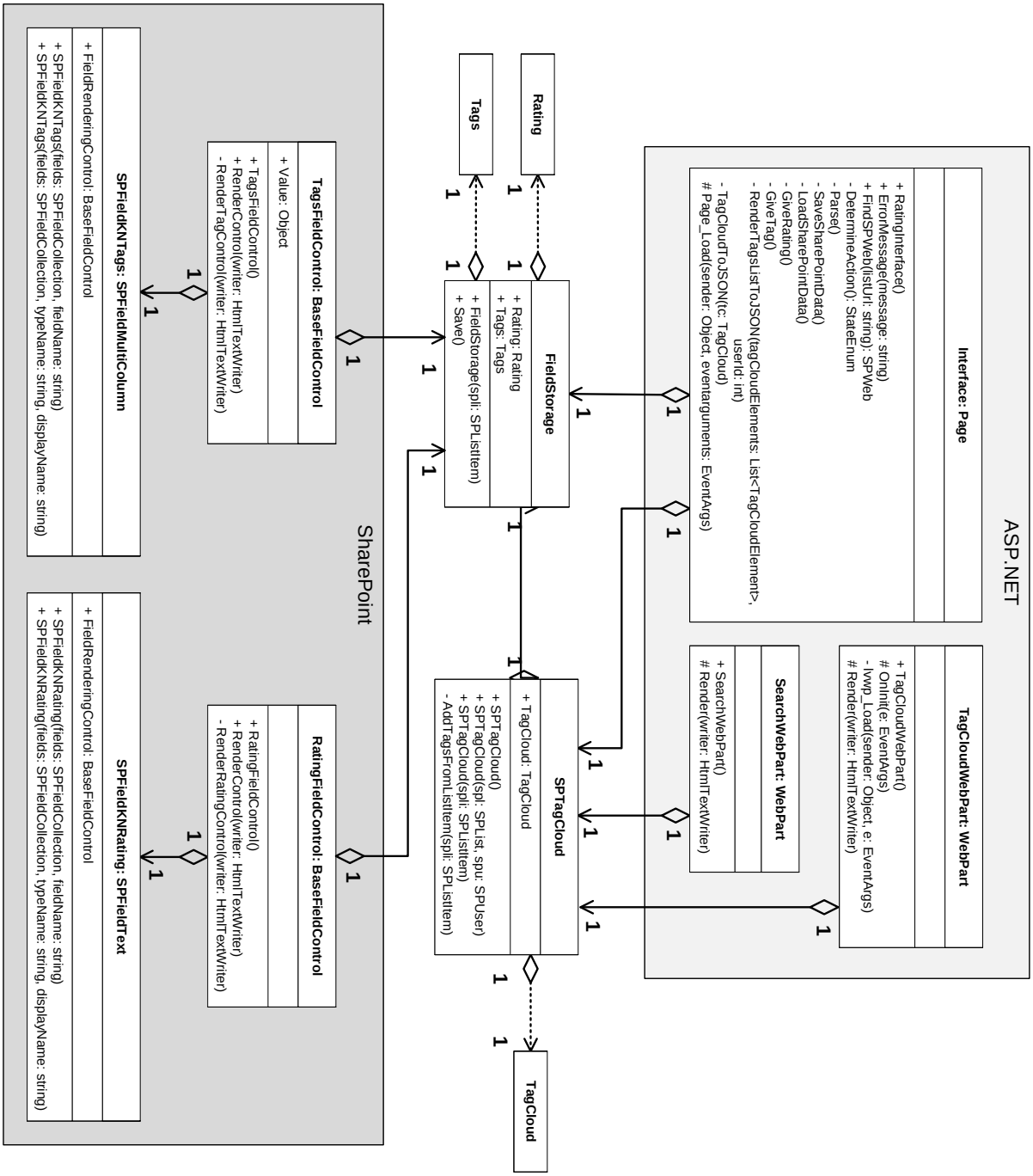
De klassendiagrammen tonen de relaties tussen de klassen in het systeem. In deze diagrammen zijn de private variabelen van de klassen weggelaten. In de klassendefinities van paragraaf 5.2 worden de private variabelen wel beschreven, ook wordt hier bij elke methode informatie gegeven over pre- en postcondities en mogelijke exceptions.

## 5.1 Klassendiagrammen

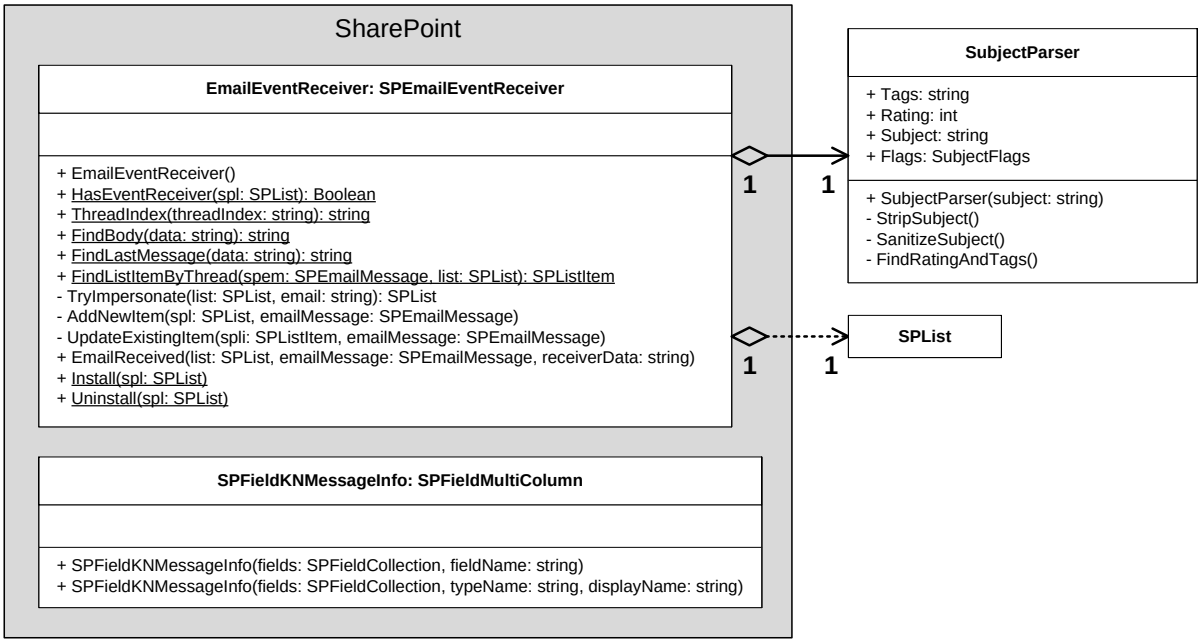
### 5.1.1 Program Logic



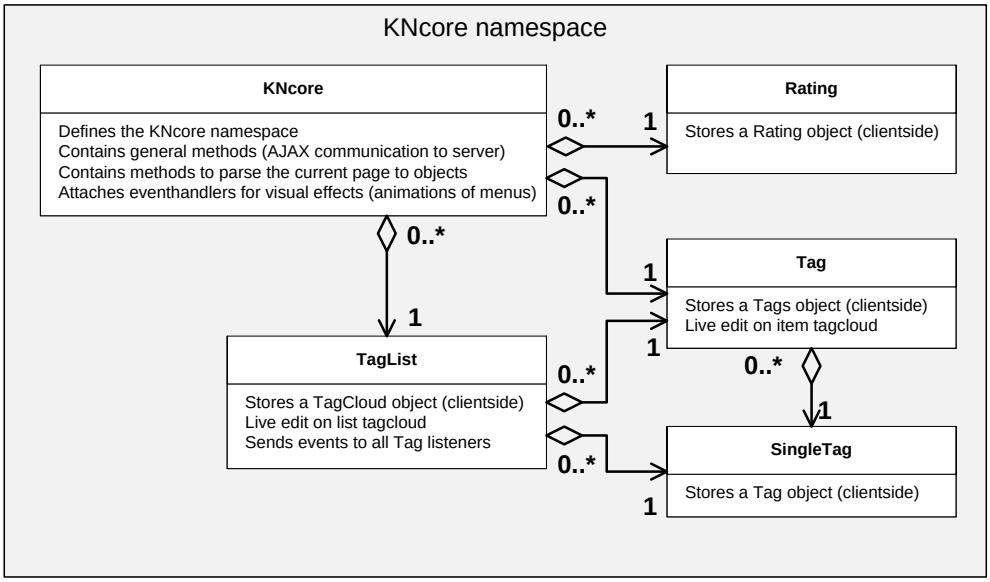
### 5.1.2 Data-interface

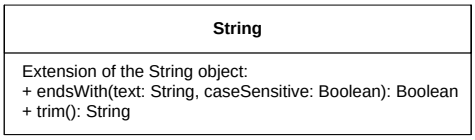


### 5.1.3 E-mail events




### 5.1.4 JavaScript



## 5.2 Klassendefinities

In deze paragraaf wordt van twee klassen (Tag en Tags) de gedetailleerde beschrijving gegeven. Per klasse is een lijst van variabelen, properties en methoden gegeven. Alle tekst is in het Engels zodat deze ook kan worden gebruikt bij het commentaar dat in de code is geplaatst.

## Tag [IComparable<Tag>]

This class represents a Tag, which is part of the Tags class. The Tag class is comparable with other Tag objects, for the order see *CompareTo*.

### Variables

**- tagName: String**

The name of a tag.

**- correctedName: String**

The corrected name of a tag, in case a spelling mistake was made (reserved for future use).

**- voteCount: int**

The total number of votes given to a tag.

**- userIds: List<int>**

A list of all user IDs from users who added a vote for a tag.

### Properties

**+ TagName: String**

Gets the name of a tag.

**+ VoteCount: int**

Gets the number of users who voted for a tag.

**+ TagStore: String**

Serializes all information about a tag. The format of the serialized string is:
`tagName:correctedName:voteCount:,(userId,)+`

### Constructor

**+ Tag(tagName: String, correctedName: String, voteCount: int, List<int>)**

Initializes a new instance of a Tag class.

#### *Parameters*

**tagName**

The name of a tag.

**correctedName**

The corrected name of a tag (used in case of spelling mistakes).

**voteCount**

The number of people who voted for a tag.

**userIds**

A list of user IDs from users who voted for a tag.

#### *Postcondition*

A new instance of a Tag is created with the given parameters.

## Methods

### + HasVoteByUser(userId: int): Boolean

Determines whether the user with ID `userId` has voted for a tag.

#### *Parameters*

**userId**

The ID of the user to check for.

#### *Return value*

`true` if the user with id `userId` voted for a tag, otherwise false.

### + AddVote(userId: int)

Adds a user vote for a tag.

#### *Parameters*

**userId**

The ID of the user who voted for a tag.

#### *Precondition*

The user did not already vote for a tag.

#### *Postcondition*

The user voted for a tag and the total number of votes is increased with one.

#### *Exceptions*

An ArgumentException is thrown when the user already voted for a tag.

### - EscapeText(text: String): String

Escape a string so that it's safe to store it in a *SPFieldMultiColumnValue*.

#### *Parameters*

**text**

The text to escape.

#### *Return value*

The escaped text, in which all occurrences of ~ (tilde), : (colon) and | (pipe) have been escaped.

### - UnescapeText(text: String): String

Unescape a string that has been escaped with *EscapeText*.

#### *Parameters*

**text**

The text to unescape.

#### *Return value*

The unescaped text. The escaped characters (~, :, |) have been restored.

**+ CompareTo(other: Tag): int**

Comparator for the IComparable interface. This makes it possible to sort a list of Tag objects. Order is based on the number of votes, from high to low. If the number of votes is equal, the tagName is used for the comparison.

#### *Parameters*

**other**

Another Tag instance to compare with a tag.

#### *Return value*

An integer that indicates the relative order of the Tag being compared.

**+ Parse(tagStore: String): Tag**

Deserializes a Tag instance stored in a string with *TagStore*.

#### *Parameters*

**tagStore**

Serialized version of a Tag instance.

#### *Return value*

A new Tag instance.

#### *Exceptions*

An ArgumentException is thrown when the string cannot be parsed.

## Tags [IEnumerable<Tag>, IEnumerable]

Represents a collection of Tag instances belonging to one item.

### Variables

**- tagsVoteCount: int**

The total number of votes for all tags in the collection.

**- tags: Dictionary<String,Tag>**

A dictionary containing all the Tag instances, accessible via the tag name.

**- topTags: List<Tag>**

A sorted list of the most popular tags.

### Properties

**+ TagsStore: String**

Serializes all tags in the collection and returns them as a string. See *Tag.TagStore* for details about the serialization format of a single Tag object. Serialization format: `|(Tag|)*`

**+ PopularTagsStore: String**

Returns the most popular tags, comma-separated, for display in the list. At most three tags are shown.

**+ TagsVoteCountStore: String**

Returns the total number of votes.

## Constructors

**+ Tags**

Creates a new instance of an empty Tags collection.

**+ Tags(tagsVoteCount: int, Dictionary<String,Tag>)**

Initializes a new instance of a Tags collection.

### *Parameters*

**tagsVoteCount**

The total number of votes for all tags.

**tags**

A dictionary containing all the Tag instances, accessible via tag name.

## Methods

**- GetEnumerator: IEnumerator<Tag>**

Returns an enumerator that iterates through a Tag collection.

### *Return value*

An *IEnumerator<Tag>* that can be used to iterate through the collection.

**- GetEnumerator: IEnumerator**

Returns an enumerator that iterates through an object collection.

### *Return value*

An *IEnumerator* that can be used to iterate through the collection.

**+ TryGetTag(tagName: String, tag: Tag&): Boolean**

Gets the Tag instance associated with the specified tag name.

### *Parameters*

**tagName**

The name of the Tag to get.

**tag**

Reference to an uninitialized Tag object.

### *Return value*

`true` if the Tags collection contains a Tag instance with the specified tag name, otherwise false.

**+ AddTag(tagName: String, userId: int)**

Adds a tag to the collection.

### *Parameters*

**tagName**

The tag name to add.

**userId**

The id of the user who adds the tag.

### *Exceptions*

An ArgumentException is thrown when an invalid tagName is specified.

**+ Parse(tagsVoteCountStore: String, tagsStore: String): Tags**

Deserializes a Tags object stored in a string with *TagsStore* and *TagsVoteCountStore*.

### *Parameters*

**tagsVoteCountStore**

Serialized version of the total tag vote count.

**tagsStore**

Serialized version of a collection of *Tag* objects.

### *Return value*

A collection of tags.

### *Exceptions*

An ArgumentException is thrown when one of the strings cannot be parsed.

# Bijlage A: Serialisatieformaat

Hier volgt het gekozen dataformaat voor de eigen veldtypen. Voor de opslag in SharePoint-kolommen wordt een multicolumn-veld gebruikt, waarbij er meerdere waarden in één kolom worden opgeslagen. Bij het opslaan en uitlezen moet de index van de subkolom worden opgeven. De gegevens uit de klassen SPFieldKNRating en SPFieldKNTags worden als volgt opgeslagen:

| Subkolom | Gegevens | Voorbeeld |
|---|---|---|
| **0** | Populaire tags | TU Delft, Tam Tam |
| **1** | Versienummer dataformaat | 1 |
| **2** | Gemiddelde rating | 4.5 |
| **3** | Aantal ratings | 2 |
| **4** | Ratings | \|1:5\|3:4\| |
| **5** | Aantal stemmen op tags | 3 |
| **6** | Tags | \|TU Delft::2:,234,456,\|Tam Tam::1:,456,\| |

### Versienummer dataformaat

Dit nummer is opgenomen om het mogelijk te maken in de toekomst met een nieuw opslagformaat te werken. Op zo'n moment wordt het versienummer opgehoogd en kan de software bij het laden van oude data herkennen dat de gegevens zijn opgeslagen in het vorige formaat, dat op een andere manier moet worden omgezet naar de interne representatie.

### Populaire tags, Gemiddelde rating, Aantal ratings, Aantal stemmen op tags

De informatie uit deze kolommen wordt tijdens de weergave van de items gebruikt om statistieken te tonen. Omdat dit in de lijstweergave gebeurt, is hier enkel CAML beschikbaar en is het efficiënter om deze gegevens op te slaan in plaats van ze te bepalen op het moment dat ze nodig zijn.

### Ratings

Van elke rating wordt het cijfer opgeslagen en het id van de gebruiker die het cijfer heeft gegeven. Dit gebeurt in het volgende formaat:

|(**userId**:**rating**|)*

### Tags

In het Tags-veld wordt een lijst van geserialiseerde Tag-objecten opgeslagen in het volgende formaat:

|(**tagName**:**correctedTagName**:**tagVoteCount**:,(**userId**,)+|)*

Omdat de tekens | en : mogen voorkomen in een tagName én worden gebruikt als scheidingsteken binnen het Tags-veld, worden deze karakters met een *escape character* opgeslagen.

| Karakter | Escape sequence |
|---|---|
| ~ | ~~ |
| : | ~@ |
| \| | ~! |

# Bijlage B: CAML

Voor het tonen van een kolom met een eigen veldtype in de lijstweergave moet CAML worden gebruikt. CAML is een zeer beperkte taal. De belangrijkste functies zijn GetVar, SetVar voor het gebruiken van variabelen, IfEqual/Switch en IfSubString voor het nemen van beslissingen. Er zijn echter geen rekenkundige operaties mogelijk, noch een groter-dan of kleiner-dan vergelijking. Om deze functionaliteit toch te gebruiken is de volgende code geschreven:

```
<SetVar Name="RatingLTELookup" Scope="Request">
	<Switch>
		<Expr><GetVar Name="Rating" /></Expr>
		<Case Value=""   ></Case><!-- Empty field, happens on create in a doclib -->
		<Case Value="0"  >;0;</Case>
		<Case Value="1"  >;0;1;</Case>
		<Case Value="1.5">;0;1;1.5;</Case>
		<Case Value="2"  >;0;1;1.5;2;</Case>
		<Case Value="2.5">;0;1;1.5;2;2.5;</Case>
		<Case Value="3"  >;0;1;1.5;2;2.5;3;</Case>
		<Case Value="3.5">;0;1;1.5;2;2.5;3;3.5;</Case>
		<Case Value="4"  >;0;1;1.5;2;2.5;3;3.5;4;</Case>
		<Case Value="4.5">;0;1;1.5;2;2.5;3;3.5;4;4.5;</Case>
		<Case Value="5"  >;0;1;1.5;2;2.5;3;3.5;4;4.5;5;</Case>
		<Default></Default><!-- Other field -->
	</Switch>
</SetVar>
```

Code snippet 2: CAML-code voor een lookup table

Deze lookup table zorgt ervoor dat er een ‘kleiner dan of gelijk aan’-conditie gebruikt kan worden op de Rating variabele met de volgende code.

```
<IfSubString>
	<Expr1><i>;</i><GetVar Name="StarCount" /><i>;</i></Expr1>
	<Expr2><GetVar Name="RatingLTELookup" /></Expr2>
	<Then>
		<!-- StarCount <= Rating -->
	</Then>
	<Else>
		<!-- StarCount > Rating -->
	</Else>
</IfSubString>
```

Code snippet 3: CAML-code voor ‘kleiner dan of gelijk aan’-conditie

Technische Universiteit Delft
Tam Tam

# Testrapport

Knowledge Network

Joost-Wim Boekesteijn
Benjamin W. Broersma
2007-11-23
V. 2.0

# Inhoudsopgave

# 1 Inleiding

Dit document beschrijft het testplan dat is gebruikt om de onderdelen van het Knowledge Network te testen. Resultaten van de tests zijn ook in dit document opgenomen. Definities van gebruikte termen zijn te vinden in het Requirements Analysis Document en het Architectural Design Document.

Tijdens het ontwikkelen van het systeem zijn er unit tests gemaakt voor de interne klassen. Deze tests controleren of de geschreven code zich houdt aan de specificaties. De verschillende typen tests zullen in dit document worden beschreven.

De functionaliteit van het systeem wordt getest aan de hand van de use cases uit het Requirements Analysis Document. Dit houdt in dat gebruikers en beheerders de taken uit de use cases succesvol kunnen uitvoeren. Daarnaast worden er tests uitgevoerd die betrekking hebben op in het Architectural Design Document genoemde criteria als veiligheid, gebruiksvriendelijkheid en snelheid.

# 2 Definities

De volgende termen worden in dit document gebruikt.

**AJAX**

*Asynchronous Javascript And XML* is een implementatietechnologie van interactieve webpagina's waarin asynchroon gevraagde gegevens worden opgehaald van de webserver. Daardoor hoeven dergelijke pagina's niet in hun geheel ververst te worden.[1]

[1] http://nl.wikipedia.org/wiki/Asynchronous_JavaScript_and_XML?oldid=9135914

# 3 Tests

Voor het testen is er onderscheid gemaakt tussen statische code-analyse, unit tests, systeemtests en gebruiksvriendelijkheidtests.

## 3.1 Statische code-analyse

Voor de code die is geschreven in JavaScript en C# is er gebruik gemaakt van statische analyse. Deze analyse is voor C# uitgevoerd met Microsoft FxCop[2] en Code Analysis van Microsoft Visual Studio 2005 Team Edition. De JavaScript-code is gecontroleerd met behulp van JSLint[3]. Statische analyse van code helpt om syntaxfouten te detecteren en herkent een aantal 'bad practices', maar kan niet controleren of de logica klopt en of klassen voldoen aan hun specificaties. Daarom zijn er op het niveau van de code ook unit tests uitgevoerd.

De resultaten van JSLint hebben in een aantal gevallen geholpen om JavaScript-fouten te verbeteren die niet werden aangegeven tijdens het uitvoeren van de code.

Voor een prototype is het niet altijd van belang dat er in eerste instantie voldaan wordt aan alle eisen die gesteld worden door FxCop. Een voorbeeld hiervan is dat FxCop aanraadt een resource file te gebruiken voor alle constante strings. Of om geen generieke excepties te vangen, maar alle exceptie-afhandeling specifiek maken. Daarom bevat de code-analyse van de uiteindelijke code van het protoype nog verschillende waarschuwingen en fouten over 'bad practices'.

## 3.2 Unit tests

Om voor de klassen die de programmalogica implementeren te controleren of de methoden in de klassen werken zoals verwacht, worden er unit tests gebruikt. De unit tests worden met NUnit[4] uitgevoerd volgens de methode van test-driven development. Op basis van de specificaties van een klasse worden er tests geschreven voor die klasse. Wanneer dit is gedaan, zullen alle tests falen. Hierna wordt de klasse zelf geschreven en worden de tests opnieuw uitgevoerd. Pas wanneer alle tests succesvol worden uitgevoerd, voldoet de klasse aan de specificaties, mits de test consistent is met de specificatie. Na het maken van wijzigingen aan de code moeten de tests weer succesvol kunnen worden uitgevoerd (regressietests). Bij voorkeur worden de tests en de code voor de klasse door verschillende personen geschreven.

De unit tests testen de klassen die data verwerken voor de tagging en rating van items. Op deze klassen zijn 59 tests uitgevoerd.

[2] FxCop, http://www.gotdotnet.com/Team/FxCop/
[3] JsLint, http://www.jslint.com/lint.html
[4] NUnit, http://www.nunit.org/

### 3.2.1 Programmalogica

De functionaliteit van de klassen wordt getest door op de data bepaalde bewerkingen uit te voeren en het formaat van de uitvoerdata te controleren. In de testcode voor de klasse Tag wordt er eerst een nieuw Tag-object gemaakt met testdata erin:

```
[SetUp]
public void Setup()
{
        userlist = new List<int>();
        userlist.Add(1);

        // create a new Tag instance with name 'TU Delft' and one vote from user with ID 1
        tag = new Tag("TU Delft", "", 1, userlist);
}
```

**Code snippet 1: Setup van de tests voor de klasse Tag.**

Hierna worden de tests uitgevoerd op dit Tag-object. Een voorbeeld is de test waarin de gebruiker met user ID 2 stemt op de tag. Na het stemmen wordt gecontroleerd of de properties en methoden van het Tag-object de verwachte waarden teruggeven. Het interne serialisatieformaat wordt ook gecontroleerd.

```
[Test]
public void AddVote()
{
        tag.AddVote(2);

        Assert.AreEqual(2, tag.VoteCount);
        Assert.IsTrue(tag.HasVoteByUser(1));
        Assert.IsTrue(tag.HasVoteByUser(2));
        Assert.IsFalse(tag.HasVoteByUser(3));
        Assert.AreEqual("TU Delft::2:,1,2,", tag.TagStore);
}
```

**Code snippet 2: Een gebruiker stemt op een tag.**

De methode *AddVote* (hier opgenomen zonder commentaar) bevat de volgende code:

```
public void AddVote(int userId)
{
        if (!userIds.Contains(userId))
        {
                userIds.Add(userId);
                voteCount++;
        }
        else
        {
                throw new ArgumentException(String.Format("Tag.AddVote: user with id {0}
                                                already voted for this tag", userId));
        }
}
```

**Code snippet 3: De geteste methode AddVote uit de klasse Tag.**

Er zijn ook gevallen waarin een exception wordt verwacht bij het uitvoeren van de test. In onderstaand voorbeeld stemt de gebruiker met user ID 1 opnieuw op het Tag-object dat is geïnitialiseerd in de methode *Setup*. Volgens het ontwerp van deze klasse moet er in dat geval een

ArgumentException worden gegooid (te zien in Code snippet 3). Dit wordt in de test met behulp van het attribuut ExpectedException[5] aangegeven.

```
[Test]
[ExpectedException(typeof(ArgumentException))]
public void UserAlreadyVoted()
{
        tag.AddVote(1);
}
```

**Code snippet 4: Gebruiker met user ID 1 stemt opnieuw op de tag.**

### 3.2.2 Serialisatie

De data van de klassen Tag, Tags en Rating wordt in een eigen opslagformaat bewaard. Elke keer dat gebruikers een tag toevoegen of een rating geven, wordt deze data door de programmacode opgehaald uit de lijst, aangepast en weer teruggeschreven. Voor dit deel van de code zijn tests geschreven die controleren of gegevens uit de lijst worden omgezet naar de correcte interne representatie en of er een foutmelding wordt gegeven wanneer de gegevens in de database niet meer in het juiste formaat staan.

De methode *Parse* in de klasse Tag zet de string-representatie van de data uit een Tag-object om naar een Tag-object. Er zijn relatief veel tests geschreven voor deze methode en soortgelijke methoden uit de klassen Tags en Rating. Enkele voorbeelden van deze tests:

```
[Test]
[ExpectedException(typeof(ArgumentException))]
public void InvalidColumnCount()
{
        Tag.Parse("::");
}
```

**Code snippet 5: Ongeldig aantal kolommen in de string.**

```
[Test]
[ExpectedException(typeof(ArgumentException))]
public void InvalidUserID()
{
        Tag.Parse("a::1:,a,");
}
```

**Code snippet 6: De waarde 'a' wordt opgegeven als user ID, terwijl hier enkel gehele positieve getallen zijn toegestaan.**

Deze twee tests verwachten allebei dat er een exception wordt gegooid door de *Parse*-methode.

### 3.2.3 SharePoint-klassen

Voor de klassen die functionaliteit van SharePoint uitbreiden zijn geen unit tests geschreven. Deze klassen zijn zeer eenvoudige wrappers, waarbij er zoveel mogelijk functionaliteit in andere klassen is geplaatst die los te testen zijn. Voor deze andere klassen zijn wel tests geschreven.

Daarnaast is het testen van SharePoint-klassen in een omgeving buiten SharePoint zeer lastig omdat niet goed gedocumenteerd is wat verschillende methoden in de basisklassen doen. Ook is de invoer en uitvoer sterk afhankelijk van het SharePoint-objectmodel en de SharePoint-context. In deze klassen wordt bijvoorbeeld gebruik gemaakt van de klasse SPContext waarmee informatie kan worden opgevraagd over de huidige SharePoint-context. Deze klasse bevat methoden en properties

[5] ExpectedExceptionAttribute, http://nunit.net/index.php?p=exception&r=2.4.4

om informatie terug te geven over de huidige website, de huidige lijst, etc. Om hier goede tests voor te schrijven, zou er een kopie moeten worden gemaakt van grote delen van het objectmodel van SharePoint. Er is daarom besloten geen unit tests voor de SharePoint-klassen te schrijven omdat een dergelijke kopie zeer groot en complex zou worden.

## 3.3 Integratietests

Tijdens de integratietests worden alle losse componenten gecombineerd en samen getest. Er is geen formele integratietest uitgevoerd in een aparte projectfase, omdat deze tests tijdens het ontwikkelen al zijn uitgevoerd. In dit project betekende het integreren dat de code binnen de SharePoint-omgeving werd uitgevoerd. Dit is tijdens het ontwikkelen vrijwel continu gedaan omdat van tevoren niet altijd bekend wat het gedrag van SharePoint zou zijn, vanwege ontbrekende of onvolledige documentatie. Door deze manier van werken hebben we besloten om geen aparte testfase te introduceren voor de integratietests.

## 3.4 Systeemtests

De systeemtests zijn uitgevoerd op het volledige systeem en niet meer op losse onderdelen zoals bij de unit tests. In het Architectural Design Document zijn veiligheid en snelheid/schaalbaarheid als ontwerpcriteria genoemd. Deze criteria zijn hier getest.

### 3.4.1 Veiligheid

De hier gebruikte veiligheidsbegrippen zijn met definitie en voorbeelden te vinden in Bijlage C: Veiligheidsdefinities en voorbeelden.

#### *3.4.1.1 Restricties*

De veiligheidstest is beperkt omdat er een aantal veiligheidsrisico's uitgesloten kunnen worden. Omdat de solution geen gebruik maakt van SQL is SQL injection niet mogelijk. Verder wordt er geen gebruik gemaakt van dynamische code evaluatie of upload mogelijkheden, hierdoor is er geen gevaar voor (remote) code execution. Daarnaast worden er geen dynamische commando's uitgevoerd op het systeem en hierdoor is command execution ook uit te sluiten.

De risico's waarop wel getest dient te worden zijn XSS, CSRF en problemen in rechtenmanagement. Onder dit laatste valt ook dat er geen informatie mag worden gelekt die normaliter via SharePoint niet te verkrijgen is (zoals of een lijst of lijstitem bestaat).

Om de interface extra te beschermen is het noodzakelijk om een specifieke HTTP-header toe te voegen om de interface te benaderen. Er kan voor testdoeleinden ook een speciale query parameter worden toegevoegd in plaats van de HTTP-header. Bij het testen is hiervan gebruik gemaakt. Deze testbeslissing is gemaakt om het systeem makkelijker te kunnen testen. In de productieversie is het niet mogelijk om deze query parameter mee te sturen.

#### *3.4.1.2 XSS*

Ondanks de genomen coderingsmaatregelen bleek het mogelijk om in de testversie met Internet Explorer 7 onder Windows 2003 JavaScript-code uit te voeren door een tag toe te voegen met JavaScript-code erin. Dit kan enkel worden misbruikt als de aanvaller een tag met scriptcode kan toevoegen en dus gebruiker van het systeem is. Vervolgens moet de aanvaller de gebruiker overhalen om de pagina te openen zodat de scriptcode wordt uitgevoerd. Hiervoor is de speciale

query parameter nodig die alleen is ingeschakeld tijdens het testen. Dit veiligheidsprobleem werd veroorzaakt door een oude fout in de detectie van het content type in Internet Explorer[6].

Verder is een probleem gevonden met het onjuist escapen van een slash. Dit zorgde niet voor veiligheidsproblemen, maar het was wel mogelijk om een fout te veroorzaken bij het ophalen van informatie over de tags van een lijstitem.

#### *3.4.1.3 CSRF*

Het testen op mogelijke CSRF komt neer op het testen of de opdrachten die worden uitgevoerd – om bijvoorbeeld een rating en tag toe te voegen – uit te voeren zijn vanaf een andere pagina dan een lijst- of detailweergave. Voor het uitvoeren van opdrachten zijn er echter een aantal parameters verplicht, zoals een lijst url en een lijstitem id. Deze zijn in principe onbekend bij niet-gebruikers, hierdoor is het zeer lastig een externe CSRF uit te voeren. Gebruikers kunnen in MOSS 2007 pagina's uploaden met JavaScript. Dit geeft de mogelijkheid om vanaf hetzelfde domein via AJAX de lijstweergave pagina op te vragen en alle opdrachten volledig te simuleren. Hierom is het enkel zinvol om mogelijke CSRF van externe domeinen te onderzoeken.

Verder is er gekeken naar CSRF risico's. Door een extra specifieke HTTP-header toe te voegen om een actie uit te voeren is het niet mogelijk om CSRF uit te voeren vanaf een extern domein. Alleen tijdens het testen wanneer ook de speciale query parameter voldoet, is het mogelijk CSRF uit te voeren. Er is echter wel een geldige lijst url en lijstitem id nodig. Het is dan mogelijk om andere gebruikers aan een lijstitem een rating of tag te laten geven zonder dat zij dit door hebben.

#### *3.4.1.4 Rechten en informatielekken*

Om het lekken van informatie te testen kan er gekeken worden of er verschillend gedrag is in het geven van rating en tags aan een niet bestaand lijstitem en een verborgen lijstitem waar geen leesrechten voor zijn. Hiertussen zou geen verschil moeten bestaan.

Er is bij het testen geen verschil geconstateerd tussen het benaderen van de interface met niet bestaande lijst url's en/of lijstitem id's en het benaderen met lijst url's en/of lijstitem id's waarvoor geen leesrechten zijn. Wel is er opgemerkt dat een deel van de interne kolomopslag zichtbaar kan worden in de Ontolica search. Microsoft lijkt deze kolom als ruwe data te indexeren zonder bij de weergave hiervan de betreffende CAML of FieldControl te gebruiken. Dit kan geen lijsten of lijstitems weergeven die normaal niet zichtbaar zijn maar wel een gebruikersnummer koppelen aan een rating of tag.

### 3.4.2 Snelheid

In het Requirements Analysis Document is bij de ontwerpcriteria de volgende eis gegeven:

> "Een lijst van 500 items met rating en tags moet niet meer dan 5 seconden trager worden weergegeven."

Omdat het aantal gegeven tags per item van invloed is op de snelheid waarmee de lijst wordt getoond, zullen er per item tussen 10 tot 20 tags worden toegevoegd. Het plan was deze test uiteindelijk op de productieserver uit te voeren, om er zeker van te zijn dat de gebruikers geen last hebben van vertraging. De snelheidstest kon echter nog niet plaatsvinden op de productieserver en

[6] MIME Type Detection in Internet Explorer, http://msdn2.microsoft.com/en-us/library/ms775147.aspx

heeft daarom op een gevirtualiseerde server plaatsgevonden. De virtuele server is door de virtualisatie langzamer dan de productieserver.

Voor de snelheidsmeting is gebruik gemaakt van WatiN[7]. Dit is een programmeerbare laag om Internet Explorer. Het biedt de mogelijkheid om via C#-code een Internet Explorer op te starten, naar een pagina te gaan en hier acties uit te voeren. Dit kon goed gebruikt worden om de snelheid te meten die een gebruiker ervaart bij het opvragen van een pagina. Het gaat dan niet alleen om de tijd die de server er over doet om een antwoord terug te sturen, maar ook de HTML-rendering en JavaScript-parsing worden meegenomen in de tijdsmeting. Dit kan omdat er letterlijk een browser wordt gebruikt om de pagina op te vragen. Vervolgens zijn er verschillende testsituaties gemaakt waar 100 keer een tijdsmeting op is uitgevoerd.

Ten eerste is er een simpele lijst A gemaakt met 500 items als referentie voor de metingen. De tweede lijst B bevatte de rating en tag kolommen, die afhankelijk van de testsituatie werden gevuld. Daarnaast bevatte lijst B een TagCloudWebPart waarin een tagcloud werd getoond met tags van alle lijstitems. De uitgevoerde metingen zijn te vinden in Bijlage D: Resultaten snelheidsmeting.

Lijst A wordt in alle 3 de metingen geladen rond de 1,4 seconde (zie Figuur 1 t/m Figuur 3). Vervolgens is lijst B gevuld met tags. Voor de namen van de tags werd een Nederlandse woordenlijst gebruikt waaruit willekeurige sets woorden werden geselecteerd. Dit resulteerde in 4968 verschillende tags op de lijst. Een test waarbij de tagcloud werd getekend is na een paar pogingen afgebroken, omdat het laden van de pagina langer dan 10 seconden duurde. Dit was ruim over de grens van 5 seconden vertraging. Vanwege deze grote snelheidsproblemen is er gekeken naar de vertraging van de verschillende onderdelen. Dit onderzoek is opgenomen in de bijlage. Het onderdeel wat voor de grootste en meest significante vertraging zorgt is het doorlopen van alle lijstitems in een TagCloudWebPart. Als er geen tags aanwezig zijn, maar het WebPart wel wordt toegevoegd is er al een zeer grote vertraging te merken (zie Figuur 12). Het WebPart doorloopt tijdens het laden alle lijstitems, in dit geval 500, ook al bevatten de lijstitems geen tags.

De conclusie kan worden getrokken dat de klasse TagCloudWebPart op een andere manier geïmplementeerd moet worden. Er zal een vorm van caching gebruikt moeten worden. Wel moet er rekening gehouden worden dat niet alle lijstitems zichtbaar zijn voor alle gebruikers. Ook kan er gekeken worden naar de CAML code, die door het gebruik van minder variabelen kan worden versneld.

## 3.5 Gebruiksvriendelijkheidtests

De tests voor gebruiksvriendelijkheid worden uitgevoerd door eindgebruikers. Voor deze tests wordt ‘hallway testing’[8] gebruikt, waarbij er door vijf willekeurig gekozen gebruikers[9] een serie van simpele taken zal worden uitgevoerd. De taken zijn gebaseerd op de scenario’s, functionele eisen en use cases uit het Requirements Analysis Document. De lijst van taken is te vinden in Bijlage A: Takenlijst acceptatietest.

De gebruiksvriendelijkheid wordt getest door de gebruikers te observeren tijdens het uitvoeren van de taken die onderdeel zijn van de functionele tests. Bij het uitvoeren van deze taken wordt er gelet

[7] WatiN, versie 1.2.0, http://watin.sourceforge.net/
[8] Joel Spolsky, http://www.joelonsoftware.com/articles/fog0000000043.html
[9] Jakob Nielsen, http://www.useit.com/alertbox/20000319.html

op de handelingen die gebruikers uitvoeren om een taak te voltooien. De resultaten zullen worden gebruikt om aanpassingen te maken aan de interface bij onderdelen waar gebruikers problemen mee hebben. Daarna zal er opnieuw worden getest.

### 3.5.1 Resultaten

Tijdens het uitvoeren van de taken zijn de gebruikers geobserveerd. Wanneer ze problemen hadden met het uitvoeren van een taak, is genoteerd wat er fout ging. Deze notities zijn te vinden in Bijlage B: Testresultaten gebruiksvriendelijkheid.

Bij bijna alle testers bleek dat de werking van het tags-menu bij de items onduidelijk was. Dit menu kan namelijk worden uitgeklapt, maar daar is geen duidelijke visuele indicatie voor. Pas wanneer het menu volledig wordt uitgeklapt, kunnen eigen tags worden toegevoegd. Ook vonden gebruikers het vervelend dat het menu ineens verdween wanneer ze hun muiscursor buiten het menu bewogen, of dat het menu juist op het scherm bleef staan wanneer ze het weg wilden hebben. Dit komt doordat het menu op twee verschillende manieren kan werken: automatisch verdwijnen wanneer de muiscursor het menu verlaat of zichtbaar blijven totdat het menu met een klik wordt gesloten. Sommige gebruikers klikten per ongeluk op de knop om het menu ‘vast te zetten’ en kregen het hierna niet meteen dicht.

Over de weergave van de tags en rating zijn ook opmerkingen gekomen: de korte teksten in het tags-menu moeten duidelijker worden gemaakt, zodat duidelijk is welke tags er precies worden getoond. Ook is de weergave van de rating niet direct duidelijk wanneer er een eigen rating is gegeven die hoger of lager is dan de gemiddelde rating. Het gaat hierbij om subtiele verschillen in de manier waarop de sterren worden getoond. Er wordt gekeken of hier een duidelijkere weergave voor kan worden gemaakt.

Verder zijn er meerdere testers geweest die het opviel dat de tagcloud onderaan de lijst niet direct werd bijgewerkt nadat ze een eigen tag hadden toegevoegd. De opmaak van de bezochte links in de tagcloud zou ook moeten worden aangepast om het verschil duidelijker te kunnen zien. Daarnaast zou de positie van de tagcloud volgens één van de testers moeten kunnen worden verplaatst, omdat deze bij een grote lijst van het scherm af kan vallen (er moet worden gescrolld).

De testers merkten ook op welke functionaliteit er nog ontbrak: het kunnen veranderen van de gegeven rating voor een item en het verwijderen van een toegekende tag aan een item. Deze opties zijn in het prototype nog niet beschikbaar, maar worden waarschijnlijk later toegevoegd.

Tijdens het testen zijn verder een aantal bij ons bekende bugs ‘ontdekt’ door onze testers, zoals de gelimiteerde weergave van zoekresultaten wanneer er wordt gesorteerd op rating. Deze bug moet in de onderliggende software worden opgelost (Ontolica/Microsoft search).

# Bijlage A: Takenlijst acceptatietest

**Document Library**

1. Voeg een document toe via de Upload-knop.
2. Geef een rating aan het document.
3. Voeg een bestaande tag toe aan het document.
4. Voeg een nieuwe tag toe aan het document.
5. Filter de lijst zodat je alleen documenten met een bepaalde tag ziet.
6. Schakel het filter uit zodat je alle documenten weer kunt zien.

**Zoeken**

1. Zoek op de term ‘design’ en sorteer de resultaten op rating.
2. Filter de zoekresultaten op de tag ‘bachelorproject’.
3. Schakel het filter op de tag weer uit zodat je alle zoekresultaten ziet.

## Bijlage B: Testresultaten gebruiksvriendelijkheid

De geanonimiseerde resultaten van de tests:

**Persoon 1:**

- Tag-menu blijft niet open staan, interface voor uitklappen onduidelijk.
- Positie tagcloud onhandig bij grote lijst? Idee: apart menu of knop voor tonen TagCloud.
- Clear filter interface is niet mooi.

**Persoon 2:**

- Problemen met uitklappen van tag-menu. Feedback nodig na toevoegen van tag?
- TagCloud is niet up-to-date na toevoegen van tag.
- Idee: tag-menu sticky laten zijn na actie van gebruiker.
- Documenten zonder rating staan niet in resultaten (Ontolica/MS-bug)

**Persoon 3:**

- Onderste deel tag-menu uitklappen is niet duidelijk aangegeven.
- Problemen met stickiness tag-menu. Wanneer wel/niet?
- TagCloud is niet up-to-date na toevoegen van tag.
- Populaire tags in de tagcloud: per … lijst/item/globaal?
- Aangeven met puntjes … wanneer er meer dan drie tags zijn bij een item (in kolom).
- Weergave rating duidelijk genoeg?
- Positie van tags + menu.

**Persoon 4:**

- Idee: open/sluit-gedrag van tag-menu veranderen: pas dicht als je er buiten klikt bv?
- Verwijderen van tags niet mogelijk?
- Case sensitivity van tags gewenst?

**Persoon 5:**

- Revote niet mogelijk?
- Nieuw scenario: aanmaken van eigen view.
- ‘Haha-bug’. Na klikken op item wilde het niet weggaan.
- CSS voor :visited toevoegen, voor duidelijkheid in TagCloud.

# Bijlage C: Veiligheidsdefinities en voorbeelden

Van een aantal aangehaalde veiligheidsbegrippen worden hier de definities genoemd en simpele praktijkvoorbeelden gegeven om te illustreren wat het begrip in essentie betekend.

## 1. Definities

**Cross-site request forgery (CSRF)**

Het ongeoorloofd laten uitvoeren van opdrachten op een andere website door de webbrowser van de gebruiker. Het bijeffect hiervan is dat de cookies en authenticatiegegevens worden meegestuurd door de webbrowser. Zo is het mogelijk om opdrachten uit te voeren namens de ingelogde gebruiker, dit wordt ook wel *session riding* genoemd.

**Cross-site scripting (XSS)**

Het zelfde bron principe zegt dat een script van de ene bron niet de pagina en gegevens, zoals cookies, van een andere bron mag lezen of wijzigen. Wanneer een aanvaller het zelfde bron principe voor HTML-scripting probeert te omzeilen spreekt men van cross-site scripting.[10]

## 2. Voorbeelden

**CSRF**

Een aanvaller neemt op zijn website http://aanvaller een afbeelding op met de verwijzing naar de overmakingsfunctie van de bank van de gebruiker http://bank/overmaken?bedrag=999&begunstigde=aanvaller. Als de gebruiker de website van de aanvaller bezoekt zal de browser de afbeelding proberen op te vragen door de verwijzing naar de bank te volgen. De webbrowser stuurt naar deze verwijzing alle authenticatie en cookie gegevens die voor dit domein zijn opgeslagen. Als de gebruiker reeds is ingelogd en de bank geen overige beveiliging bezit zal de betaling geschieden.

Dit is een zeer simpel voorbeeld, dat in het geval van banken zeer onrealistisch is omdat deze vaak meerdere stappen vereisen voordat een overboeking wordt uitgevoerd. Cross-site request forgery is redelijk onbekend en wordt vaak niet goed begrepen. Veel websites zijn kwetsbaar voor dergelijke aanvallen.

**XSS**

Een dynamische website gebruikt een parameter uit het adres van de website om een bericht op de pagina te schrijven. Een voorbeeld hiervan is http://website/pagina?bericht=test, waarbij test wordt uitgeschreven op de pagina. Als het bericht niet wordt gecodeerd, kan er HTML injectie plaatsvinden. Een aanvaller kan dan JavaScript opnemen op de pagina, dat vervolgens de cookies van de gebruiker kan uitlezen en de pagina kan manipuleren.

[10] http://nl.wikipedia.org/wiki/XSS?oldid=8464302

# Bijlage D: Resultaten snelheidsmeting

De rating en tags zijn leeg als er niet specifiek is aangegeven dat deze er wel zijn.

Voor het onderzoek naar de snelheid per onderdeel zijn in lijst B alle tags verwijderd. Enkel de rendering van de kolom met CAML en JavaScript-parsing zorgen voor ruim 1,5 seconde vertraging ten opzichte van lijst A (zie Figuur 4).

Als de JavaScript voor de speciale kolommen niet wordt meegerekend scheelt dit ongeveer 0,6 seconde (zie Figuur 5). Het uitvoeren van de JavaScript-code gebeurt op de achtergrond en de pagina is te bekijken en gebruiken zonder JavaScript. De extra functionaliteit wordt pas bruikbaar als de JavaScript is uitgevoerd.

Lijst B heeft geen verschil in laadtijden bij verschillende pagina's (zie Figuur 6).

Wanneer er aan lijst B bij elk lijstitem 100 ratings (van andere gebruikers) worden toegekend verslechtert de laadtijd niet (zie Figuur 7). Opvallend is dat de laadtijd significant wordt verkort wanneer hier een eigen rating van 3 sterren aan wordt toegevoegd (zie Figuur 8). Het blijkt hieruit dat de CAML hiervoor sneller is, er hoeft nu geen rating link te worden opgebouwd waar relatief veel variabelen voor worden gebruikt. Het is opmerkelijk dat het gebruik van meer variabelen zorgt voor een vertraging tijdens het uitvoeren van de code.

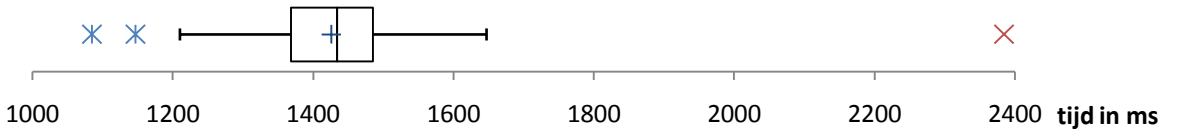
Van tevoren werd er wel verwacht dat een eigen rating van 1 ster trager zou laden dan een rating van 3 sterren. Dit omdat CAML voor de detectie van 1 ster 5 in plaats van 1 "IfSubString" operaties moet uitvoeren. Dit is nodig door de beperkingen van CAML, de enige manier om te detecteren welke rating is gegeven is door alle mogelijkheden één voor één te testen. Er is echter geen duidelijke vertraging door deze extra operaties meetbaar (zie Figuur 8 en Figuur 9). Wel wordt de JavaScript code van de pagina versneld als er zelf gestemd is (zie Figuur 10). Dit komt omdat enkel JavaScript wordt gebruikt bij lijstitems waar een rating aan kan worden gegeven.

Als er bij elk lijstitem een willekeurig aantal tags wordt gevoegd, wordt er een kleine vertraging opgemerkt (zie Figuur 11). Dit komt waarschijnlijk door de grote hoeveelheid extra data die de lijst dan bevat.

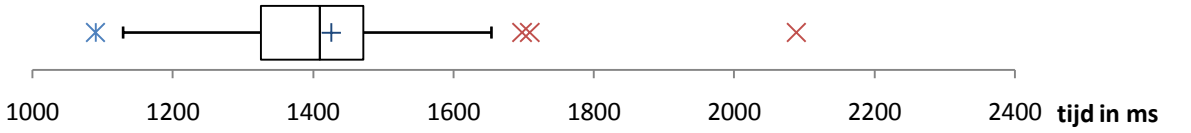
Vervolgens is een TagCloudWebPart toegevoegd op lijst B met leeggemaakte tags en ratings. Als er geen tags aanwezig zijn is er al een zeer grote vertraging te merken (zie Figuur 12). De tagcloud doorloopt tijdens het laden alle lijstitems, in dit geval 500, ook al bevatten de lijstitems geen tags.

Bij de afgebroken test waarbij bijna vijfduizend verschillende tags werden toegevoegd moet worden opgemerkt dat, naast de trage serverkant, de rendertijd van al deze tags in de HTML in de browser niet te verwaarlozen is.

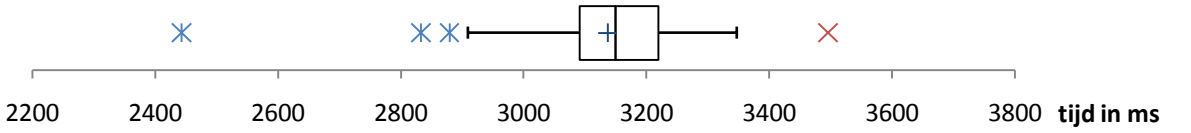

Figuur 1: Boxplot snelheid Lijst A




Figuur 2: Boxplot snelheid Lijst A na lijst B vulling

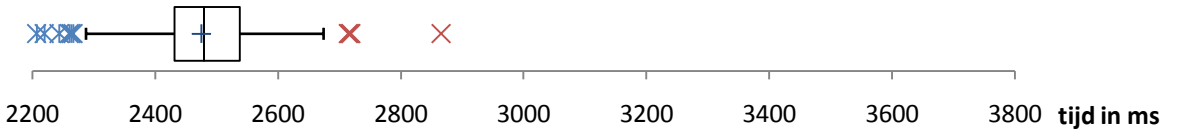



Figuur 3: Boxplot snelheid Lijst A, tweede pagina

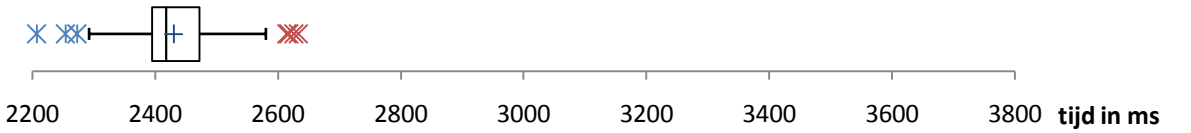



Figuur 4: Boxplot snelheid Lijst B, met JS, zonder TagCloud

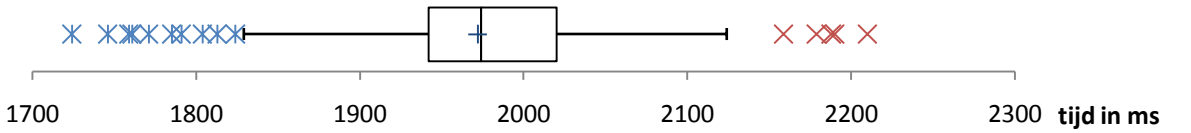

Figuur 5: Boxplot snelheid Lijst B, zonder JS, zonder TagCloud



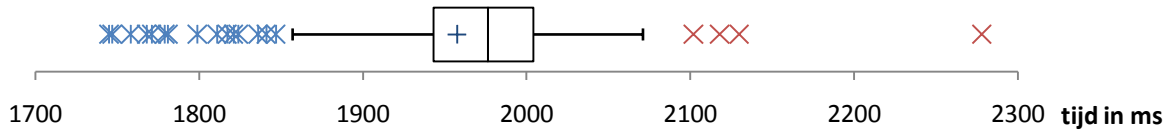

Figuur 6: Boxplot snelheid Lijst B, zonder JS, zonder TagCloud, tweede pagina



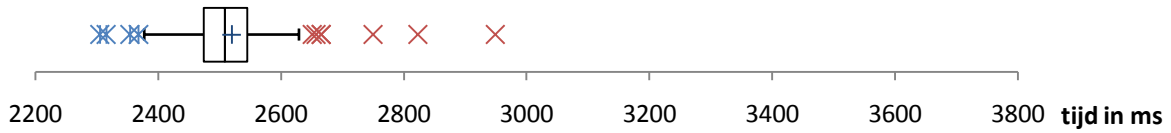

Figuur 7: Boxplot snelheid Lijst B, zonder JS, zonder TagCloud, met rating zonder eigen rating



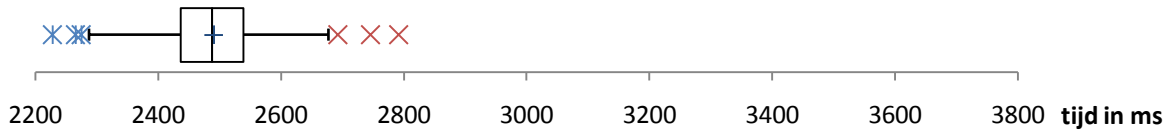

Figuur 8: Boxplot snelheid Lijst B, zonder JS, zonder TagCloud, met rating waarbij eigen rating van 3



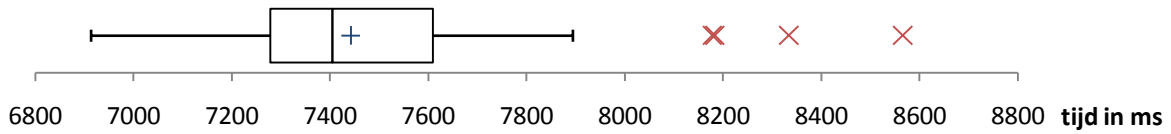

Figuur 9: Boxplot snelheid Lijst B, zonder JS, zonder TagCloud, met rating waarbij eigen rating van 1




Figuur 10: Boxplot snelheid Lijst B, met JS, zonder TagCloud, met rating waarbij eigen rating van 1

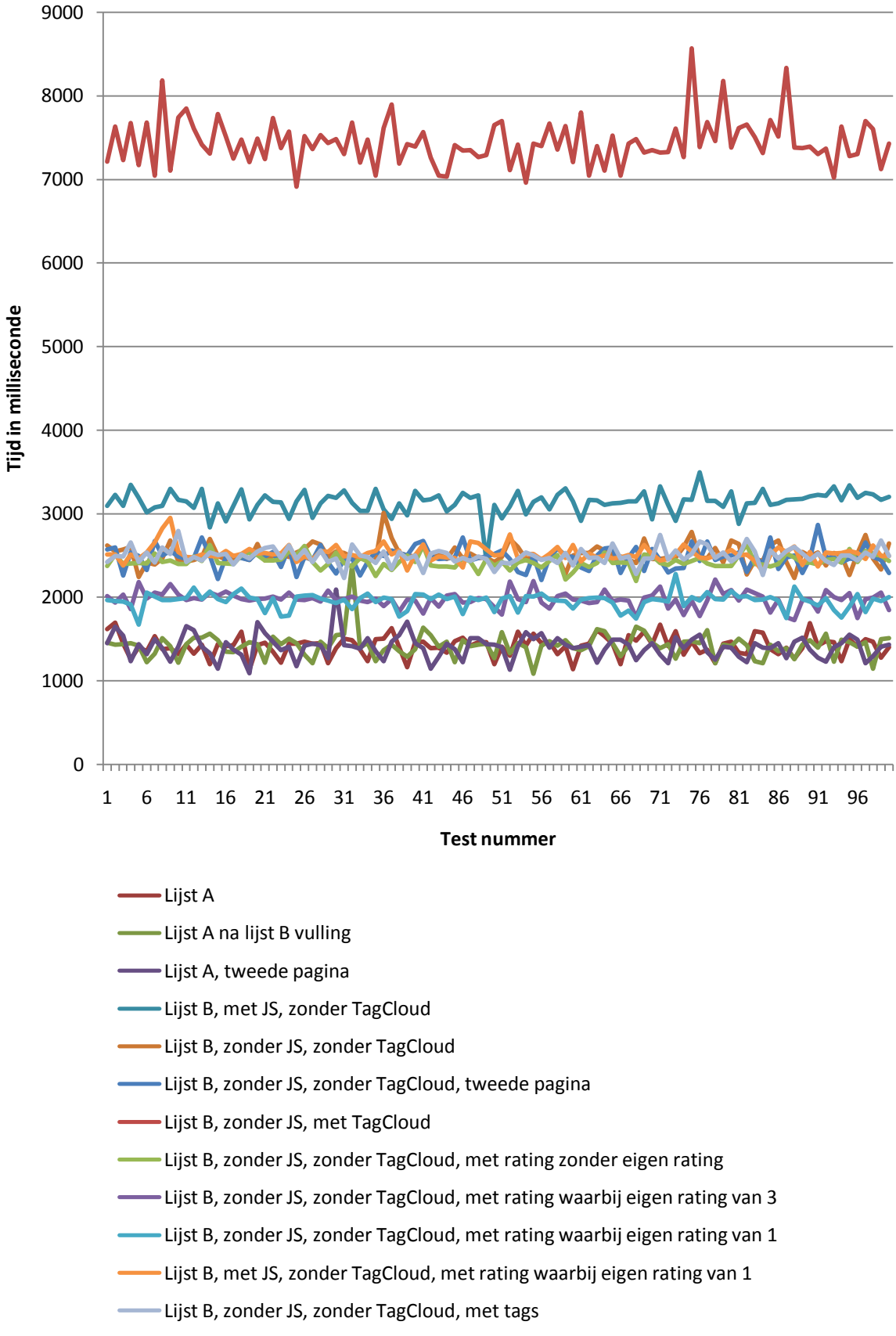



Figuur 11: Boxplot snelheid Lijst B, zonder JS, zonder TagCloud, met tags




Figuur 12: Boxplot snelheid Lijst B, zonder JS, met TagCloud

Hieronder nog een overzicht van alle onafhankelijke metingen in één overzicht.




Figuur 13: 12 onafhankelijke snelheidstesten met 100 metingen gecombineerd in 1 overzicht