Vrije Universiteit Brussel

FACULTEIT INGENIEURSWETENSCHAPPEN & WETENSCHAPPEN


# Software Project Management Plan

Software Engineering

Nicolas Carraggi, Youri Coppens, Christophe Gaethofs, Pieter Meiresone, Sam Van den Vonder, Fernando Suarez, Tim Witters

Academiejaar 2013-2014

# Versie geschiedenis

**Tabel 1:** Versie geschiedenis

| Versie | Datum | Auteur | Commentaar |
|---|---|---|---|
| 0.1 | 4/11/2013 | Pieter Meiresone | Initiële versie |
| 1.0 | 15/11/2013 | Pieter Meiresone | Iteratie 0 |

# Inhoudsopgave

# Lijst van figuren

# Lijst van tabellen

# Hoofdstuk 1

# Overzicht

## 1.1 Samenvatting van het project

### 1.1.1 Doel, scope en objectieven

Het doel van dit project is het maken van een webapplicatie die het mogelijk maakt om lessenroosters binnen de universiteit te schedulen. Deze lessenroosters moeten vervolgens door de studenten geraadpleegd kunnen worden. Er is verder specifieke support voor mobiele platformen nodig.

Als projectnaam is gekozen voor "CalZone", geïspireerd op het feit dat we een zone voor een kalender moeten maken. Het logo is afgebeeld in figuur 1.1.


**Figuur 1.1:** Het logo.

### 1.1.2 Project deliverables

De deliverables voor dit project zijn de volgende:

- Functionerende website die voldoet aan de requirements gespecifieerd in het SRS.
- Software Project Management Plan (SPMP)
- Software Test Plan (STD)
- Software Requirements Specification (SRS)

- Software Design Document (SDD)
- Minutes van alle vergaderingen
- Source code en aanverwanten

Deze documenten worden in .pdf formaat samen in een enkele zipfile verstuurd via mail. De documenten zijn ook beschikbaar op de GitHub repository. Op het einde van elke iteratie wordt ook de code van het project opgeleverd, deze zit dan in dezelfde zipfile als de documenten. In tabel 1.1 staat de lijst van data waarop de code en de documenten worden opgeleverd. Na elke iteratie volgt een presentatie waar het geleverde werk voor die iteratie wordt gedemonstreert en besproken.

**Tabel 1.1:** Kalender

| Datum | To Do |
|---|---|
| Maandag 04/11/2013 | Inleveren SPMP |
| Vrijdag 15/11/2013 | Eerste versie documenten |
| Vrijdag 13/12/2013 | Einde iteratie 1: opleveren code en documenten |
| Woensdag 18/12/2013 | Presentatie |
| Dinsdag 04/03/2014 | Einde iteratie 2: opleveren code en documenten |
| Woensdag 12/03/2014 | Presentatie |
| Dinsdag 15/04/2014 | Einde iteratie 3: opleveren code en documenten |
| Woensdag 23/04/2014 | Presentatie |
| Vrijdag 16/05/2014 | Einde iteratie 4: opleveren code en documenten |
| Woensdag 21/05/2014 | Finale presentatie |

Meer gedetailleerde informatie is te vinden in paragraaf 5.3.5.

## 1.2 Evolutie van het SPMP

De evoluatie van de SPMP zal bijgehouden worden met behulp van een versie geschiedenis in het begin van dit document. Er zullen steeds geplande updates uitgevoerd worden op de tijdsstippen beschreven in tabel 1.1.

Deze updates worden uitgevoerd met behulp van de GitHub Repository [3]. Met behulp van GitHub kunnen we met issues werken, en hiervoor telkens een verantwoordelijke aanduiden.

# Hoofdstuk 2

# Referenties


[1] *Portal Team Website.* http://wilma.vub.ac.be/~se2_1314/website/

[2] *ShareLateX.* https://www.sharelatex.com

[3] *GitHub Repository.* https://github.com/CalZoneVUB

[4] *GitHub API.* http://developer.github.com/v3/

[5] *Wilma.* http://wilma.vub.ac.be/

[6] *Microsft Project.* http://office.microsoft.com/nl-be/project/

[7] *Microsoft DreamSpark for VUB-Engineering Students.* http://e5.onthehub.com/WebStore/Welcome.aspx?ws=4ec25e81-649b-e011-969d-0030487d8897&vsro=8

[8] *WBS Chart Pro.* http://www.criticaltools.com/wbsmain.htm

[9] *Eclipse Metrics Plugin.* http://eclipse-metrics.sourceforge.net/

[10] *De huisstijl van de VUB.* http://www.vub.ac.be/home/huisstijl/

[11] *COCOMO I.* http://en.wikipedia.org/wiki/COCOMO

[12] *COCOMO II.* http://csse.usc.edu/tools/COCOMOII.php

[13] *Software Engineering: Modern Approaches.* Eric J. Braude & Michael E. Bernstein

[14] *Eclipse.* http://www.eclipse.org

[15] *Javadoc.* http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html

[16] *Henderson-Seller metriek.* http://eclipse-metrics.sourceforge.net/descriptions/pages/cohesion/HendersonSellers.html

[17] *Code Conventions for the Java Programming Language* http://www.oracle.com/technetwork/java/codeconv-138413.html

[18] *Javadoc Conventions.* http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

# Hoofdstuk 3

# Definities

**Tabel 3.1:** Overzicht van de gebruikte acroniemen.

| Acroniem | Betekenis |
|---|---|
| SPMP | Software Project Management Plan |
| SRS | Software Requirements Specification |
| STD | Software Test Plan |
| SDD | Software Design Document |
| SQAP | Software Quality Assurance Plan |
| SCMP | Software Configuration Management Plan |

# Hoofdstuk 4

# Project organisatie

## 4.1 Externe interfaces

Dit project wordt geproduceerd in opdracht van het vak Software Engineering. Communicatie verloopt via mail met Jens Nicolay[1], Ragnild van Der Straeten[2] en Dirk van Deun[3]. Jens Nicolay en Ragnild van Der Straeten worden gecontacteerd voor functionale zaken, terwijl Dirk van Deun gecontacteerd wordt voor technische zaken betreffende de infrastructuur.

## 4.2 Interne structuur

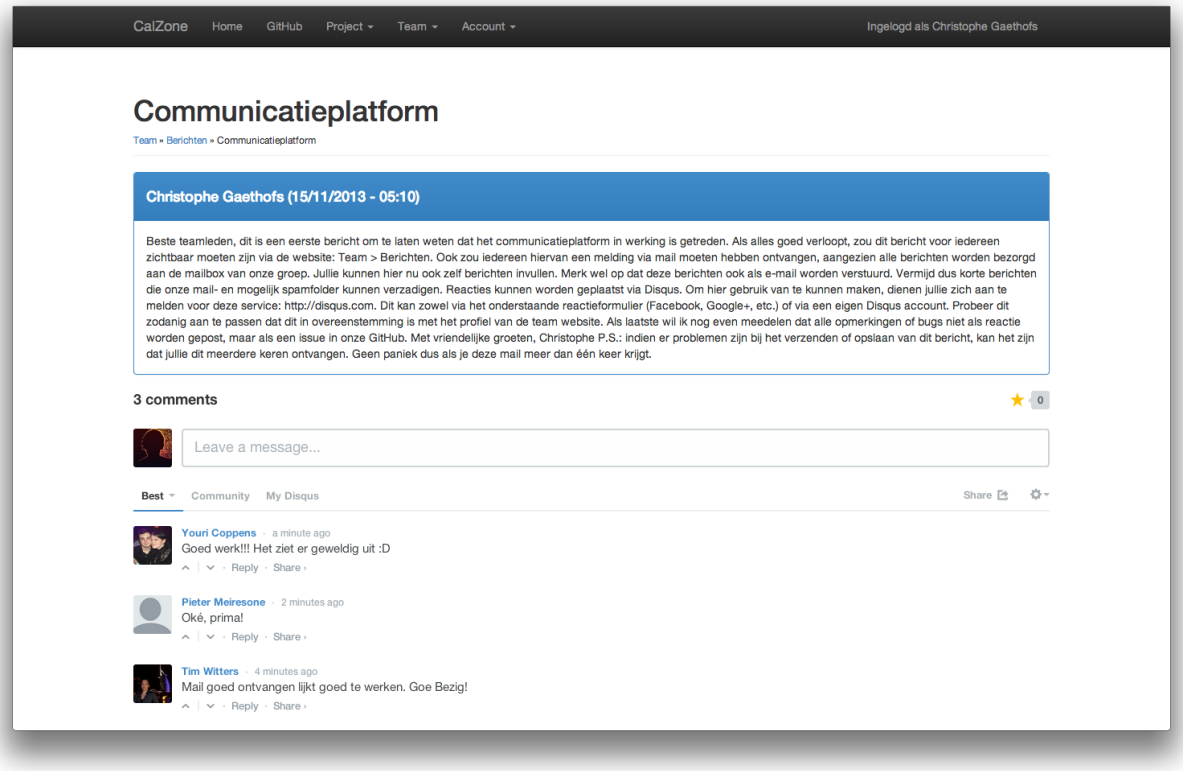
Voor dit project is er een team van 7 personen. De takenverdeling binnen dit team is als volgt:

**Tabel 4.1:** Takenverdeling.

| Rol | Verantwoordelijke | Reserve |
|---|---|---|
| Project Manager | Pieter | Nicolas |
| Configuration Manager | Christophe | Tim |
| Database Manager | Nicolas | Christophe |
| Quality assurance Manager | Sam | Youri |
| Requirements Manager | Fernando | Pieter |
| Design Manager | Youri | Sam |
| Implementation Manager | Tim | Fernando |
| Webmaster | Christophe | \ |
| Secretaris | Fernando | \ |

Communicatie binnen het team verloopt via de website [1]. Hier wordt een soort blog gebruikt door de teamleden. Een teamlid kan een bericht plaatsen en achteraf kan op dit bericht gereageerd worden door andere teamleden. Een voorbeeldinteractie is weergegeven in figuur 4.1. Verder wordt er gebruik gemaakt van de mailinglijst op wilma om de verschillende teamleden van een notificatie te voorzien bij de aanmaak van een nieuw bericht.

[1] jnicolay@vub.ac.be
[2] rvdstrae@vub.ac.be
[3] dirk@dinf.vub.ac.be




**Figuur 4.1:** Communicatie op de team website.

## 4.3 Rollen en verantwoordelijkheden

Elk teamlid is verantwoordelijk voor een functie en reserve van een andere functie (zie tabel 4.1).Ook speelt elk teamlid een rol in de codering van het systeem.

Hier volgt een gedetailleerd overzicht van deze functies:

**Tabel 4.2:** Functies.

| Functie | Verantwoordelijkheden |
|---|---|
| Project Manager | • Verantwoordelijkheid over SPMP<br>• Opvolging van managers<br>• Zorgt ervoor dat deadlines binnen het team gerespecteerd worden<br>• Ingrijpen waar nodig |
| Configuration Manager | • Verantwoordelijkheid over SCMP (onderdeel van SPMP)<br>• Keuze software en procedures<br>• Controle van gebruik van software en instellingen. |
| Database Manager | • Onderhoud en controle database<br>• Onderscheid maken tussen en beheren van test en officiële database |
| Quality Assurance Manager | • Verantwoordelijkheid over SQAP (onderdeel van SPMP)<br>• Verantwoordelijkheid over STD<br>• Controle en correctie van nauwkeurigheid en styling van code en documenten<br>• Algemene testing van software |
| Requirements Manager | • Verantwoordelijkheid over SRS<br>• Controle van uitvoering van requirements<br>• Verificatie van uitgewerkte requirements |
| Design Manager | • Verantwoordelijkheid over SDD<br>• modelleren en architectuur bepalen van het systeem |

**Tabel 4.3:** Functies (vervolg).

| Functie | Verantwoordelijkheden |
|---|---|
| Implementation Manager | • Verdelen van system requirements onder de developers<br>• Controle van de progressie van de code<br>• Controle van de uitvoering van het design |
| Webmaster | • Onderhoud teamwebsite<br>• Onderhoud projectwebsite (die dat de gebruikers van het systeem gebruiken). |
| Secretaris | • Verslagen van vergaderingen bijhouden |

# Hoofdstuk 5

# Management process

## 5.1 Project start plan

Voor het project management zal er gebruikt worden van Microsoft Project [6]. Een licentie valt gratis te verkrijgen via Microsoft Dreamspark for VUB Students [7]. M.b.v. Microsoft Project zullen er Gantt charts gegenereerd worden. Voor Work Breakdown Structures zal er gebruik worden gemaaakt van WBS Chart Pro [8]. Hiervan is een demo-versie verkrijgbaar die voldoende functionaliteit biedt. WBS Chart Pro kan op basis van een Microsoft Project file onmiddelijk een WBS genereren.

In deze paragraaf bespreken we de verwachte kost van het project. Voor de kostberekening maken we gebruik van het COCOMO I model [11]. Ondanks dat het COCOMO I model reeds verouderd is t.o.v. COCOMO II [12], verkiezen we toch COCOMO I. De reden hiervoor is dat COCOMO II afhant van veel inputparameters. Vermits we hiervoor weinig tot geen ervaring hebben, verkiezen we COCOMO I dat slechts afhangt van 2 inputparameters. Bij COCOMO I zijn de vrije parameters het project type en aantal lijnen code. Als type project moeten we kiezen tussen "simple", "semidetached" en "embedded". We kiezen hierin voor het type "semidetached". Dit weerspiegelt goed de huidige situatie van ons team. De omvang van ons team is aan de grote kant en er zijn veel onbekende factoren in dit project (zie ook 5.4). We berekenen dan de effort $E$ uitgedrukt in persoonsmaanden (pm) als

$$E = a * KLOC^b$$

Hierbij zijn $a$ en $b$ constanten gegeven door het COCOMO I-model. Voor ons is het interessanter om naar de werkuren te kijken i.p.v. het aantal persoonsmaanden. Volgens de COCOMO standaard bevat één werkmaand 152 uren. De benodigde tijd ($T$) is dan

$$T = 152 \frac{u}{pm} * E$$

Hierbij wordt T uitgedrukt in uren. Uit de tabellen van het COCOMO-model volgt dat $a = 3.0$ en $b = 1.12$. Verder schatten we in dat dit project een totale omvang zal hebben van 10KLOC. Dit geeft dan:

$$T = 152 * 3.0 * 10^{1.12} \approx 6011u$$

Vermits ons team uit 7 personen bestaat, geeft dit per persoon een tijdsduur van:

$$T_{persoon} \approx 859u$$

Het is evident dat een werklast van $859u$ per persoon niet haalbaar is. Dit hoge resultaat is waarschijnlijk het gevolg van de onbetrouwbare methode COCOMO I. Tegenwoordig zijn er meer tools beschikbaar (waaronder GitHub) die de productiviteit sterk verhogen.

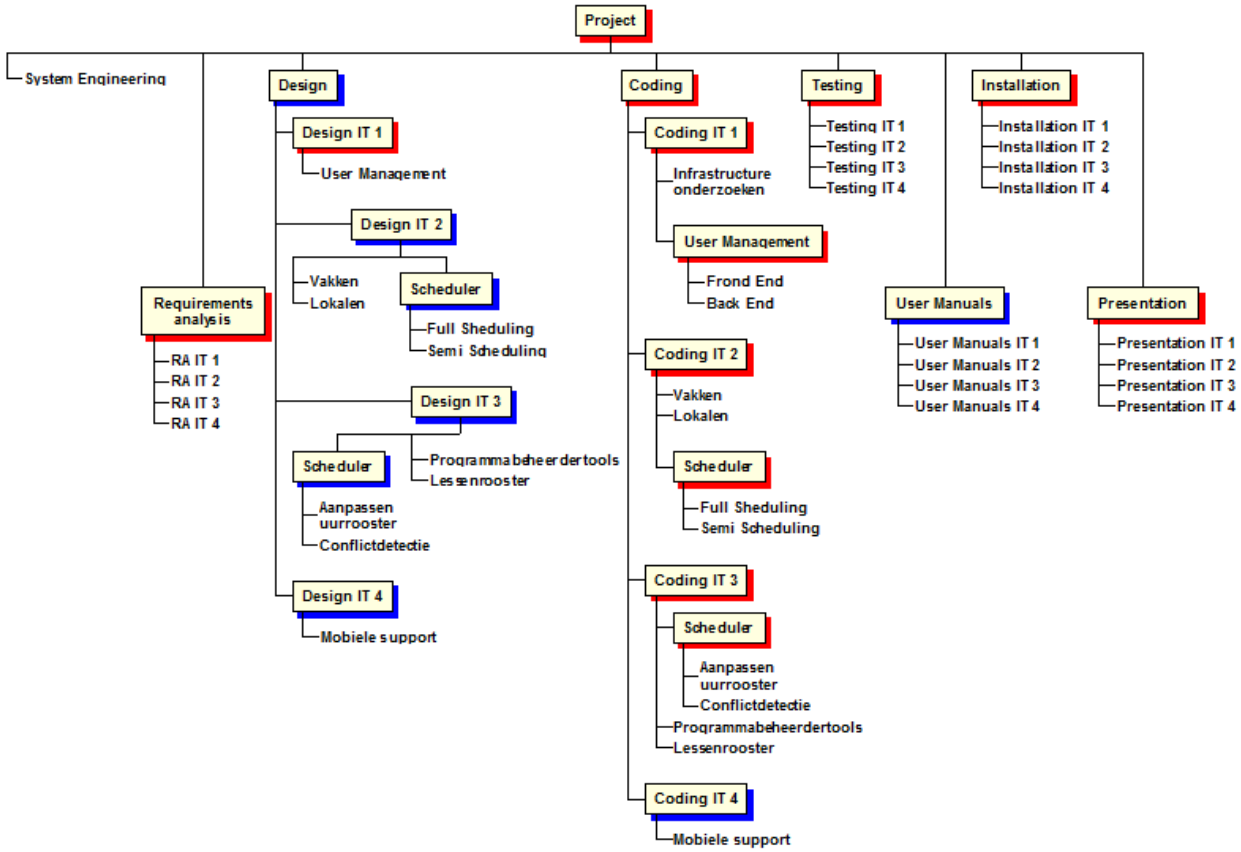
Op basis van de Gantt-chart die zal opgesteld worden in paragraaf 5.2 bekomen we een meer realistische schatting van

$$T_{persoon} \approx 300u$$

## 5.2 Werk plan

### 5.2.1 Activiteiten

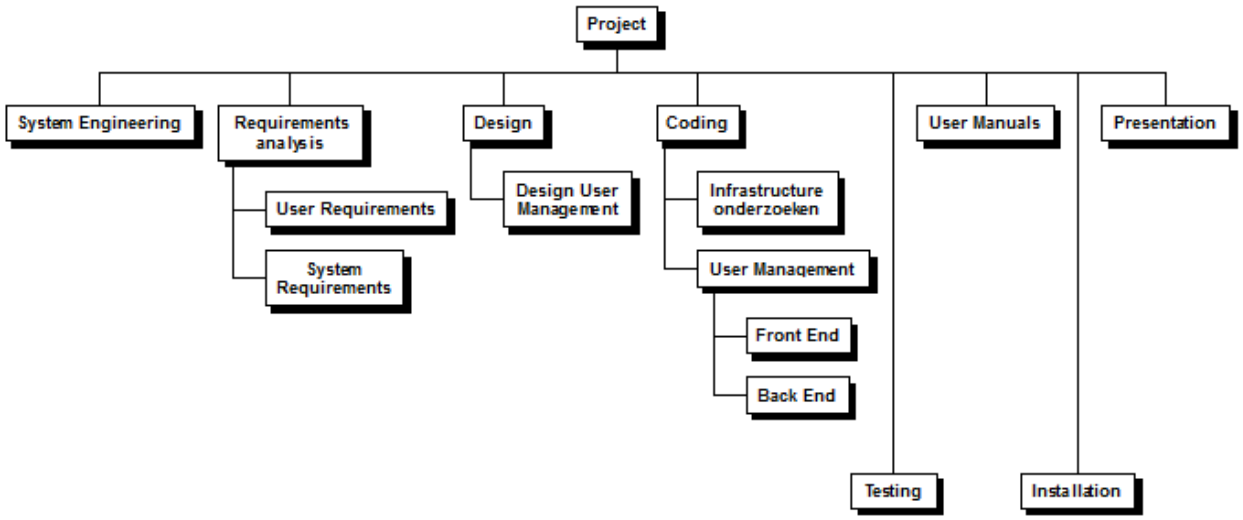
Het project bestaat uit volgende activiteiten[1], weergegeven als een work breakdown structure in figuur 5.1. De activiteiten in de WBS komen overeen met de gegroepeerde requirements die in het SRS-document en website te vinden zijn.




**Figuur 5.1:** Work breakdown structure.

Voor de eerste iteratie is de work breakdown structure weergegeven in figuur 5.2.

[1] In volgende versies van het SPMP komen hier nog verscheidene activiteiten bij.




**Figuur 5.2:** Work breakdown structure van iteratie 1.

In de volgende versies van de SPMP zullen de work breakdown structures van de volgende iteraties worden weergegeven.

### 5.2.2 Planning

In tabel 5.1 is een overzicht weergegeven van de verschillende activiteiten gedurende iteratie 1. Hierbij zijn ook de afhankelijkheden weergegeven tussen deze activiteiten. Op basis van deze tabel kunnen we een Gantt chart opstellen en het kritisch pad bepalen.

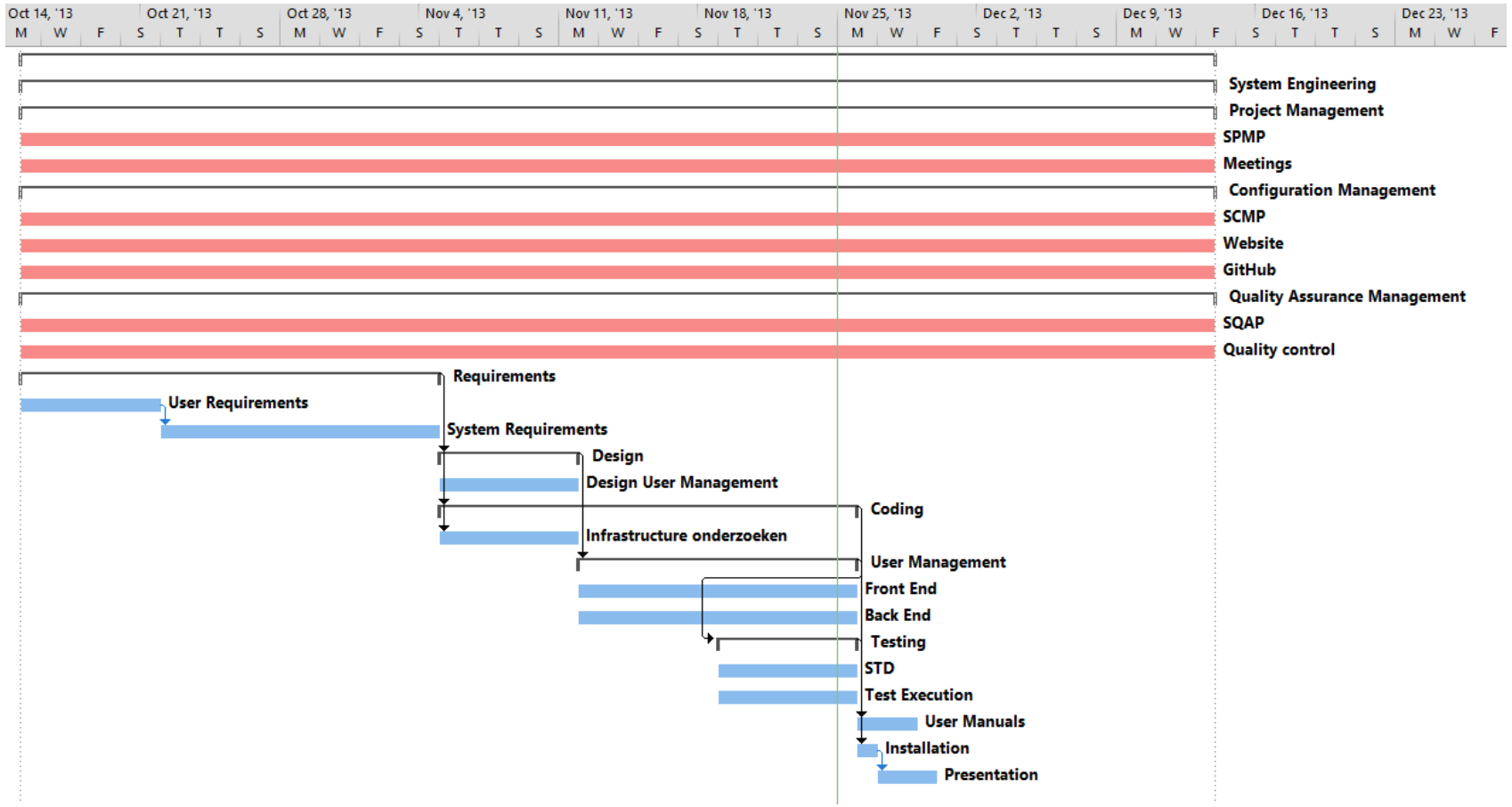
**Tabel 5.1:** Activiteiten van de eerste iteratie en afhankelijkheden.

| Activiteit ID | Activiteit Naam | Tijdsduur (in weken) | Geschatte aantal werkuren (in u) | Afhankelijkheden |
|---|---|---|---|---|
| 1 | System Engineering | n.v.t. | n.v.t. | |
| 1.1 | Project Management | n.v.t. | n.v.t. | |
| 1.1.1 | SPMP | n.v.t. | n.v.t. | |
| 1.1.2 | Meetings | n.v.t. | n.v.t. | |
| 1.2 | Configuration Management | n.v.t. | n.v.t. | |
| 1.2.1 | SCMP | n.v.t. | n.v.t. | |
| 1.2.2 | Website | n.v.t. | n.v.t. | |
| 1.3 | Quality Assurance Management | n.v.t. | n.v.t. | |
| 1.3.1 | SQAP | n.v.t. | n.v.t. | |
| 1.3.2 | Quality Control | n.v.t. | n.v.t. | |
| 2 | Requirements Analysis | | 25 | |
| 2.1 | User Requirements | 1 | 10 | |
| 2.2 | System Requirements | 2 | 15 | 2.1 |
| 3 | Design | | 20 | 2 |
| 3.1 | Design User Management | 1 | 20 | |
| 4 | Coding | | 60 | |
| 4.1 | Infrastructure onderzoeken | 1 | 15 | 2 |
| 4.2 | User Management | | 45 | 3, 4.1 |
| 4.2.1 | Frond End | 2 | 15 | |
| 4.2.2 | Back End | 2 | 30 | |
| 5 | Testing | 1 | 30 | 4.2 - 7 dagen |
| 5.1 | STD | | 15 | |
| 5.2 | Test Execution | | 15 | |
| 6 | User manuals | 3 | 8 | 4 |
| 7 | Installation | 1 | 4 | 5,6 |
| 8 | Presentation | 3 | 8 | 7 |

Op basis van tabel 5.1 bekomen we de Gantt chart voor iteratie 1 weergegeven in figuur 5.3. Het kritisch pad is weergegeven in het rood. Hierbij is nog rekening met extra constraints:

- Het project is gestart op 15 oktober 2013.
- De activiteit "Requirements Analysis" kan niet vroeger beginnen dan 24 oktober 2013.
- De activiteit "Installation" mag niet later eindigen dan 13 december 2013 (zie tabel 1.1).
- De activiteit "Presentation" mag niet later eindigen dan 18 december 2013 (zie tabel 1.1).

Op figuur 5.3 is er te zien dat er nog enkele dagen marge zijn op het einde van iteratie 1. Het kritisch pad mag dus enkele dagen vertraging oplopen.




**Figuur 5.3:** Gantt chart voor iteratie 1.

## 5.3 Controle plan

### 5.3.1 Requirements controle

Er is een requirements dashboard beschikbaar dat een overzicht weergeeft van alle requirements en hun status per iteratie (done, busy, planned, deferred, ... ). Hierbij worden telkens de belangrijkste statistieken per requirement weergegeven (percentage afgewerkt indien bezig, duurtijd implementatie indien klaar, aantal unit tests, ... ). Het onderhoud van het requirements dashboard wordt verzorgd door de Configuration Manager.

Op de website van het team [1] is er een pagina beschikbaar die het mogelijk maakt om zaken te rapporteren, en veranderingen door te voeren met betrekkeing tot de SRS. Het is aan de Requirements Manager om ervoor te zorgen dat de verschillende requirements-ID's stabiel blijven. Indien nodig overlegt de Requirements Manager met de Project Manager over eventuele invloed op de planning wanneer er wijzigingen plaatsvinden aan het SRS.

### 5.3.2 Planning controle

Voor het opvolgen en schatten van de planning wordt gebruik gemaakt van milestones op GitHub. Voor elke milestone wordt een verantwoordelijke aangesteld. De vooruitgang t.o.v. elke deadline zal gecontroleerd worden door de bijhorende verantwoordelijke en de Project Manager. Bovendien zal ook de progressie wekelijks op de vergadering besproken worden. Indien de actuele vooruitgang niet overeenkomt met de geplande vooruitgang, kunnen volgende maatregelen genomen worden:

- Planning herbekijken.
- Teams herindelen.
- Sancties voor de verantwoordelijke indien de actuele vooruitgang lager ligt door nalatigheid. Sancties worden vastgelegd door de rest van het team.

### 5.3.3 Budget controle

Op de website van het team [1] wordt er gebruik gemaakt worden van een time tracking tool die het mogelijk maakt een gedetailleerd logboek bij te houden van de reeds uitgevoerde activiteiten. Op basis hiervan kunnen we dan de "kost" berekenen van het project.

De tijdsregistratie wordt uitgevoerd bij het beëndigen van elke werkdag. Hierbij wordt telkens opgegeven aan welke activiteit men gewerkt heeft (een overzicht van de verschillende activiteiten bevindt zich in sectie 5.2).

### 5.3.4 Kwaliteitscontrole

Ook de kwaliteit zal opgevolgd worden met behulp van de website. De Quality Assurance Manager zal verantwoordelijk zijn voor de kwaliteitspagina op de website.

### 5.3.5 Rapportering

In tabel 1.1 worden de deliverables voor dit project weergegeven. Hierrond worden volgende afspraken gemaakt:

- Alle documenten en source code (inclusief unit tests) worden per mail aangeleverd als een enkele zipfile, met als naam se2-iterM, waarbij M het nummer van de iteratie is (voor eerste versie van documenten geldt $M = 0$). De aanlevering gebeurt ten laatste voor 9u00 's ochtends op de dag van de deadline (zie tabel 1.1).
- Alle documenten en source code worden worden overeenkomstig getagd/gebranchd (se2-iterM) in de GitHub repository.

- Andere artefacten (zoals executables) worden apart aangeleverd (direct, of via een link, in de opleveringsmail) en vermelden duidelijk de overeenkomstige iteratie in de bestandsnaam.
- De mail van de oplevering bevat een bondig overzicht (lijstje) van wat er precies opgeleverd wordt.

Voor het verspreiden van de resultaten zal ook gebruik worden gemaakt van de website.

- Opgeleverde documenten, source code en andere artefacten moeten publiekelijk en overzichtelijk beschikbaar zijn.
- Het opleveren van documenten en code per iteratie houdt in dat ten laatste op die welbepaalde dag (zie tabel 1.1) de site ook up-to-date wordt gebracht.

Een presentatie duurt een half uur per groep en wordt ingevuld door 2 sprekers. Alle groepsleden moeten minimum één keer presenteren. De volgende zaken worden besproken of gedemonstreerd:

- een demo van de toegevoegde functionaliteit ten opzichte van de vorige iteratie
- analyse van de ontmoete obstakels en de genomen beslissingen
- bespreking van de functionaliteiten die aan bod zullen komen in de volgende iteratie
- bespreking van eventuele obstakels, risico's, etc. in de volgende iteratie
- overzicht van de architectuur en design van de applicatie
- bespreking van de statistieken zoals de tijd per taak en per persoon en van de eventuele vertragingen (plus oplossingen om deze zo klein mogelijk te houden en te vermijden in de toekomst)

### 5.3.6 Metriek verzamelingsplan

Metrieken zullen verzameld worden met behulp van de Eclipse Metrics Plugin [9]. Er zullen metrieken op methodenniveau en klassenniveau verzameld worden. Op methodenniveau verzamelen we volgende metrieken:

1. Cyclomatic Complexity.
2. Aantal statements
3. Aantal levels.
4. Aantal lokale variabelen in de scope.
5. Aantal parameters.
6. Feature envy

Op klassenniveau verzamelen we volgende metrieken:

1. Efferent Couplings.
2. Aantal attributen.
3. Complexiteit
4. Cohesie tussen de verschillende methodes.

Een gedetailleerde uitleg over deze metrieken is te vinden in bijlage A.

De Eclipse Metrics Plugin verzamelt automatisch metrieken bij elk compile process (eens geconfigureerd). Deze worden dan geexporteerd naar een XML-file die vervolgens wordt ingelezen door de website.

Verzamelde metrieken zullen op de webpagina van het team gevisualiseerd worden. De metrieken worden verzameld tijdens de activiteiten waar er gecodeerd wordt. Bij de start van het coderen zullen er wekelijks metrieken verzameld worden. In de laatste week van coderen worden er dagelijks metrieken verzameld, hierdoor kan er refactoring plaatsvinden die de kwaliteit van de code verhoogt.

## 5.4 Risico management plan

In deze paragraaf zullen de verschillende risico's verbonden aan dit project besproken worden. In paragraaf 5.4.4 zullen deze risico's geprioritiseerd worden. Om de risico's te kunnen prioritiseren zullen we gebruiken maken van 3 parameters:

- De kans $p$ waarmee dit risico kan voorkomen. Hierbij is $p \in \{1, 2, \ldots, 10\}$. Verder betekent $p = 1$ dat het risico niet kan voorkomen en $p = 10$ dat het risico met zekerheid voorkomt.
- De impact $i$ op het project wanneer het risico werkelijkheid wordt. Hierbij is $i \in \{1, 2, \ldots, 10\}$. Verder betekent $i = 1$ een impact op het project die minimaal is en $i = 10$ een impact die maximaal is.
- De kost $c$ die het risico heeft op het project om het probleem op te lossen. Hierbij is $c \in \{1, 2, \ldots, 10\}$. Hierbij betekent $c = 1$ een lage kostprijs, terwijl $c = 10$ een hoge kostprijs betekent. Bij een hoge kostprijs zal de prioriteit van het risico lager gesteld worden, vermits het dan beter kan zijn om pas het risico weg te werken wanneer het voorkomt. Hiermee worden grote onnodige kosten vermeden.

### 5.4.1 Project risico's

**Niet-realistische Planning**

In paragraaf 5.1 is de werkduur geschat met het COCOMO I-model. Hier werd reeds duidelijk dat de geschatte tijdsduur enorm hoog ligt. Dit zorgt voor een grote druk op de planning.

Om de vooropgestelde deadlines (zie tabel 1.1) te bereiken dient men dus voldoende tijd vrij te maken. Het zwaartepunt van het academiejaar van de verschillende groepsleden ligt bij de meeste groepsleden in het eerste semester. Het grootste risico in verband met planning ligt dus vooral bij de eerste iteratie. Dit kan het best vermeden worden door gebruik te maken van interne deadlines en de progressie op te volgen tijdens de wekelijkske teammeetings. Dit is besproken in paragraaf 5.3.2.

We karakteriseren dit risico m.b.v. volgende parameters:

$$p = 8$$
$$i = 8$$
$$c = 2$$

**Communicatieprobleem**

Bij een team van zeven personen is communicatie essentieel. Indien er geen bijzondere maatregelen worden genomen, zal deze snel verkeerd lopen.

Als oplossing hiervoor wordt er reeds een mailinglijst aangeleverd door VUB. Omdat wij deze niet overzichtelijk genoeg vinden, hebben wij zelf een communicatieplatform ontwikkeld op onze website. De werking verloopt als een blog waarbij de teamleaden een bericht kunnen plaatsen en vervolgens hierop kunnen reageren. Bij elk nieuw bericht worden de teamleden van een notificatie voorzien. Dit is ook besproken in paragraaf 4.2.

De implementatie hiervan is niet triviaal en vraagt de nodige investering. Wij verwachten echter dat de "return on investment" groot zal zijn. Samengevat:

$$p = 10$$
$$i = 8$$
$$c = 8$$

**Meningsverschillen**

Bij het werken in teamverband is het onvermijdelijk dat er meningsverschillen optreden tussen verscheidene groepsleden. We maken een onderscheid tussen volgende menigsverschillen:

- Functioneel: Meningsveschillen waarvan de uitkomst bepalend is voor de uiteindelijke werking van het programma.
- Niet-functioneel: Meningsverschillen waarvan de uitkomst niet bepalend is voor de uiteindelijke werking van het programma. Bijvoorbeeld de communicatiemethode, meningsveschillen over interne deadlines, persoonlijke conflicten tussen teamleden, ...

Verder kunnen we ook nog onderscheid maken tussen kleine en grote meningsverschillen. Al deze types worden opgelost op een verschillende manier, weergegeven in tabel 5.2. De 'kost' voor het oplossen van het risico is dus minimaal.

**Tabel 5.2:** Oplossingen voor de verschillende meningsconflicten.

| | Functioneel | Niet-functioneel |
|---|---|---|
| Groot | Meeting met team, vervolgens stemming | Behandeld door de verantwoordelijke van dit onderwerp |
| Klein | Project Manager | Behandeld door de verantwoordelijke van dit onderwerp |

We karakteriseren dit risico m.b.v. volgende parameters:

$$p = 10$$
$$i = 3$$
$$c = 2$$

### 5.4.2 Technische risico's

**Gebrek aan ervaring in de implementatie-technologie**

Hiervoor wordt er gekeken naar de programmeertalen die gebruikt worden tijdens dit project (zie sectie 6.2). Een overzicht van de aanwezige kwaliteiten is zichtbaar in tabel 5.3. Wat opvalt is dat er van elks kwaliteiten aanwezig zijn, maar er zijn ook de nodige aandachtspunten. Zo is bijvoorbeeld de ervaring in Java en JavaScript beperkt.

**Tabel 5.3:** Ervaring van de verschillende teamleden.

| | Java | JavaScript | HTML \CSS | SQL | Opmerkingen |
|---|---|---|---|---|---|
| Christophe | - | ++ | ++ | ++ | Reeds ervaring opgedaan in de bedrijfswereld |
| Youri | + | - | - | ++ | Voorkeur voor logica, A.I. en modeleren. |
| Nicolas | + | - | + | ++ | Voorkeur voor back end |
| Tim | - | - | + | ++ | Reeds ervaring in C++ en andere programmeerprojecten |
| Sam | + | - | + | ++ | |
| Fernando | - | - | + | ++ | Voorkeur voor design |
| Pieter | ++ | + | + | + | Reeds ervaring opgedaan in de bedrijfswereld |

Omdat de implementatie-technologie opgelegd wordt, is het gebruik maken van andere frameworks, programmeertalen, ... geen optie. Hierdoor zullen we gebruik maken van workshops. Hiermee gaan we de ervaringen van

de verschillende teamleden op elkaar overbrengen. Een workshop wordt georganiseerd door een teamlid die zijn kennis en ervaringen over een bepaalt framework, programmeertaal, ... uiteenzet gedurende 30 à 60 minuten. Volgende workshops zijn reeds gepland:

- GitHub (door Christophe)
- Java (door Pieter)

Doordat we gebruik maken van korte workshops, zou de invloed van deze workshops op de planning minimaal zijn. We karakteriseren dit risico m.b.v. volgende parameters:

$$p = 8$$
$$i = 8$$
$$c = 4$$

#### Gebrekkige performantie

Het hoofddoel van dit project is het maken van een scheduler. Het is uiteraard gewenst dat het schedulen vlot verloopt. Vanwege de complexiteit van dit onderwerp zal er ook de nodige aandacht besteedt moeten worden aan de performantie van de scheduler.

Dit zal gemonitored worden met behulp van benchmarks. Indien er tekortkomingen ontdekt worden, zullen de nodige optimalisaties moeten doorgevoerd worden aan de scheduler. We karakteriseren dit risico m.b.v. volgende parameters:

$$p = 7$$
$$i = 2$$
$$c = 7$$

#### Support voor meerdere browsers

Uit ervaring weten we dat Internet Explorer soms voor compatibiliteitsproblemen zorgt. Het is niet nodig om dit risico op voorhand te elimineren. Op voorhand alle compabiliteitsproblemen opzoeken bij verschillende browsers zou een immens werk zijn. Wanneer bij het testen duidelijk wordt dat er problemen optreden in een bepaalde browser, moet dit opgelost worden d.m.v. de benodigde documentatie op te zoeken op internet. We karakteriseren dit risico m.b.v. volgende parameters:

$$p = 7$$
$$i = 4$$
$$c = 10$$

#### Merge conflicten

Bij het werken met de GitHub repository zijn merge conflicten nooit verweg. Door het werken met forking en pull requests vermijden we echter deze conflicten en zorgen we voor een goed beheer van onze GitHub repository. Het implementeren van forking en pull requests vergt echter enig opzoekwerk waardoor de kost hoog ligt. Het beheer van de GitHub repository wordt verder besproken in het SCMP (zie paragraaf 7.1).

$$p = 10$$
$$i = 5$$
$$c = 7$$

#### Bugs

Bugs zijn onvermijdelijk in een programma. De procedure voor bugs af te handelen wordt besproken in het problem resolution plan (zie paragraaf 7.5). Dit risico wordt gekarakteriseerd met volgende parameters:

$$p = 10$$
$$i = 7$$
$$c = 3$$

### 5.4.3 Bedrijfsrisico's

#### Ontwikkelen van de verkeerde functionaliteit

Het ontwikkelen van verkeerde functionaliteit is steeds een reëel risico. Doordat we gebruik maken van een iteratief development process, waarbij we bij elke iteratie werkende code opleveren, krijgen we geregeld feedback van de klant. Hierdoor kunnen we de requirements, indien nodig, bijsturen. Vermits we bij de start van elke iteratie een gedetailleerde planning opstellen, kunnen eventuele wijzigingen vlot verwerkt worden. We karakteriseren dit risico m.b.v. volgende parameters:

$$p = 6$$
$$i = 7$$
$$c = 4$$

### 5.4.4 Prioriteit van de verschillende risico's.

Op basis van de 3 parameters $p$, $i$ en $c$ berekenen we de prioriteit $P$ van het risico [13]

$$P = (11 - p) * (11 - i) * c$$

Vermits hoge waarden van $p$, $i$ en kleine $c$ belangrijker zijn, zijn de risico's met de kleinste waarden van $P$ het belangrijkste. Een (gesorteerd) overzicht is weergegeven in tabel 5.4.

**Tabel 5.4:** Prioriteit van de verschillende risico's.

| Risico | $p$ | $i$ | $c$ | $P$ |
|---|---|---|---|---|
| Bugs | 10 | 7 | 3 | 12 |
| Meningsverschillen | 10 | 3 | 2 | 16 |
| Niet-realistische planning | 8 | 8 | 2 | 18 |
| Communicatieprobleem | 10 | 8 | 8 | 24 |
| Gebrek aan ervaring | 8 | 8 | 4 | 36 |
| Merge conflicten | 10 | 5 | 7 | 42 |
| Verkeerde functionaliteit | 6 | 7 | 4 | 80 |
| Performantie | 7 | 2 | 7 | 252 |
| Browser support | 7 | 4 | 10 | 280 |

# Hoofdstuk 6

# Technisch process plan

## 6.1 Process model

Vanwege de opgelegde deadlines (zie tabel 1.1), zullen we gebruik maken van een iteratief model voor het opleveren van de documenten en code. Hierbij zal er steeds geittereerd worden over requirements analyse, design, constructie, testing en installatie.

## 6.2 Methodes, tools and technieken

Voor dit project zullen we enkel gebruik maken van Java, JavaScript, HTML, CSS en SQL als programmeertaal. Andere bijhorende open-source frameworks en bibliotheken kunnen ook gebruikt worden. Voor testen te schrijven zullen we gebruik maken van het JUnit framework. Dit alles zal gebeuren in Eclipse.

Er zal gebruik worden gemaakt van een public repository op GitHub [3] voor het verzamelen van de code. Verder zal er ook een repository voorzien zijn voor de verschillende documenten. Deze repositories zijn ook gekoppeld met onze website.

## 6.3 Infrastructuur plan

Voor dit project zullen we gebruik maken van de wilma server van de Vrije Universiteit Brussel [5]. Deze server bevat een mysql database. Verder wordt er gebruik gemaakt van het netwerk van de VUB.

# Hoofdstuk 7

# Supporting process plans

## 7.1 Configuration management plan

In dit onderdeel van het SPMP wordt het Software Configuration Management Plan, of kortweg SCMP, kort besproken. Een apart document voor het SCMP is voor dit project overbodig aangezien meerdere onderwerpen reeds uitvoerig in andere delen van dit document aan bod komen. Evidente onderdelen van het SCMP, zoals een beschrijving van het software project, zullen dan ook worden weggelaten.

### 7.1.1 Introductie

Het Software Configuration Management Plan heeft als doel een gestructureerd overzicht te creëren waarin wordt beschreven op welke manier er met de software wordt omgegaan, hoe deze wordt gebruikt, hoe deze in gebruik wordt gecontroleerd en hoe de werking van het team wordt gestuurd binnen bepaalde gebruiksnormen.

Dit onderdeel van dit document is bedoeld als richtlijn voor het gebruik van de software binnen het project en is gericht aan de teamleden die meewerken aan dit project, aan de Configuration Manager die deze richtlijnen dient te implementeren, te controleren en dient in te grijpen indien nodig, alsook aan derden die op deze manier de structuur en interne configuratie van werken kunnen volgen.

Omdat het gebruik van software en/of systemen stap per stap wordt geadopteerd, zal een betere vertrouwdheid hiermee een duidelijker beeld vormen over de voor- en nadelen. Het is dan ook vanzelfsprekend dat dit document, alsook de hierin beschreven systemen en software, mee zal evolueren naarmate dit nodig zal zijn. Dit telkens met oog op de vergemakkelijking van de samenwerking, verbetering van de communicatie en de verhoging van de productiviteit.

### 7.1.2 SCM Management en verantwoordelijkheden

Dit deel van het document beschrijft de allocatie van de verantwoordelijkheden en machtigingen voor de verscheidene SCM activiteiten, en het beheer hiervan.

De in dit onderdeel beschreven richtlijnen voor het gebruik van de aangeboden en verworven tools, zijn van toepassing voor elk teamlid. Het is de verantwoordelijkheid van deze leden om zich hiermee vertrouwd te maken en deze toe te passen binnen dit project. Met problemen of vragen over de gebruikte software, kunnen zij terecht bij de Configuration Manager: Christophe Gaethofs.

Het is de taak van de Configuration Manager, Christophe Gaethofs, om hulp te verlenen aan de andere teamleden, toezicht te houden dat de teamleden deze richtlijnen volgen, alsook hen op de hoogte te brengen over veranderen of bijsturingen gemaakt aan het SCMP, hetzij door een collectief of individueel genomen besluit. Slechts na

unanieme goedkeuring, zullen bijsturingen effectief worden geïmplementeerd en opgenomen in het SCMP.

Verder zal de Configuration Manager verantwoordelijk zijn voor de configuratie en controle van al de in sectie 7.1.3 opgenomen activiteiten.

Voor deze opdracht zal de Configuration Manager, Christophe Gaethofs, bijgestaan worden door Tim Witters, die de taak van Configuration Manager op zich neemt bij afwezigheid van Christophe Gaethofs, of waneer de situatie dit vereist. Dit op voorwaarde dat deze hiervan op tijd op de hoogte wordt gebracht, om hem de mogelijkheid te geven zich voor eventuele taken in te werken.

### 7.1.3 SCM Activiteiten

Dit deel van het document identificeert alle functies en taken die nodig zijn om de configuratie van het beschreven software project en bijhorende tools te beheren. Hierbij wordt rekening gehouden met de bij het project opgelegde procedures voor het indienen en beheren. Voor de controle, het beheer en het overzicht van alle activiteiten, zal voornamelijk de team website [1] en GitHub gebruikt worden.

Alle gebruikte en in dit document aangehaalde tools, die een API ter beschikking hebben en bijdragen aan de workflow, zullen in deze website worden geïntegreerd.

#### Website

De team website zal intensief gebruikt worden als platform dat voor alle leden ter beschikken wordt gesteld ter collaboratie. Het zal onder meer worden gebruikt om een overzicht te creëeren voor:

- updates, gemaakt aan het SRS of updates aan andere documenten;
- gedetailleerde informatie van alle activiteiten, beschreven in het Project Plan;
- opvolging van de vooruitgang (o.a. vooruitgang van de requirements);
- opvolging van de besteedde tijd per activiteit per lid via Time Tracking;
- de communicatie door correspondentie op te lijsten in een overzicht;
- code-overzicht, door volledige integratie van GitHub.

De website wordt ook gebruikt als tool om een eenvoudig inzicht te geven in de individuele bijdrages door de leden, aan de hand van alle ingevoerde gegevens.

Hoewel dit in eerste instantie meer werk met zich meebrengt, zal dit werk zich snel vertalingen in een betere samenwerking waarbij communicatie, duidelijkheid en overzicht primeert. Zo geeft het de teamleider en andere verantwoordelijken gecentraliseerde toegang tot alle informatie om eventuele bijsturingen van teamleden, manier van werken, programma's, ... snel te kunnen laten gebeuren.

De implementatie, onderhoud en controle van de website is de verantwoordelijkheid van de Webmaster en Configuration Manager. Beide taken zullen worden vervuld door Christophe Gaethofs. Bij eventuele conflicten of problemen op de website, dienen de leden deze hiervan dan ook zo snel mogelijk op de hoogte te brengen om deze problemen te kunnen oplossen.

De communicatie op de website zal verlopen via een berichtenpagina, waarop leden mededelingen en berichten kunnen posten. Dit systeem werkt als een blog waarbij het gepubliceerde artikel, in deze context dan bericht, direct wordt weergegeven op de website. Dit bericht wordt, vanaf het adres van het lid dat deze publicatie maakte, ook als e-mail verstuurd naar de projectmailbox. Alle leden krijgen dit bericht dan in hun inbox, maar kunnen dit ook online bekijken. De communicatiepagina's stellen hen ook in staat online te reageren op deze berichten via

Disqus (http://disqus.com/). Deze open source tool geeft een website de mogelijkheid om een reactieformulier aan een pagina toe te voegen en dit te beheren. Elke berichtpagina zal onder het bericht zo'n formulier bevatten. De geplaatste reacties worden niet via mail verstuurd.

De website is gebouwd op het opensource CMS (Content Management System) SilverStripe (http://www.silverstripe.org) en maakt gebruikt van het Bootstrap JS (http://getbootstrap.com/) framework voor het thema en JavaScript functionaliteit. Voor de kalender is een eigen module ontworpen op basis van Bootstrap Calendar (http://bootstrap-calendar.azurewebsites.net), een opensource kalender voor Bootstrap JS.

### Documenten en source code

Om alle fasen van het project ordelijk en gestructureerd te kunnen laten verlopen, zal er gebruikt gemaakt worden van publieke repositories, aangeboden door GitHub.

De configuratie hiervan dient in lijn te zijn met de opgegeven voorwaarden waardoor alle documenten en source code overeenkomstig zullen worden getagd/gebranchd (se2-iterN waarbij N staat voor het iteratienummer) in de repository.

Voor de documentatie en de source code worden verschillende repositories worden aangemaakt.

De in de repository ondergebrachte documenten, worden in Latex geschreven en bijgehouden in branches op de Git. Hierdoor zijn de stabiele documenten voor alle leden beschikbaar, worden alle wijzigingen en versies bijgehouden en kan er simultaan aan hetzelfde document worden gewerkt zonder conflicten.

Om problemen bij het mergen van branches te vermijden en controle te houden op de gemaakte wijzigingen op de Git, zal voor het coderen gebruikt worden gemaakt van forks. Dit zijn clones van de eigenlijke team repository in de persoonlijke repositories van de leden. Hierin kan men verder werken zonder conflicten te veroorzaken. Als een lid wijzigingen wil doorvoeren, dient deze een pull request uit te voeren naar de team repository. De Configuration Manager, Christophe Gaethofs, zal dan samen met de reserve Configuration Manager, Tim Witters, al deze pull requests behandelen en doorvoeren indien deze geen conflicten veroorzaken.

Alle source code is terug te vinden in de Git, toegankelijk via de website. Als editor zal er gebruikt worden gemaakt van twee open source git editors. Voor een eenvoudige werking zonder te veel verwarrende functionaliteit, zal gebruik gemaakt worden van een GitHub client. Voor de meer geavanceerde functies en het beheer van de repositories, zullen de Configuration Managers, gebruik maken van SourceTree http://www.sourcetreeapp.com.

Er wordt nooit online in de Git zelf gewerkt, tenzij bepaalde goed gegronde redenen lokaal werken niet mogelijk maken en andere teamleden hiermee akkoord gaan.

In elke fase van het project wordt er ook onderling besproken hoe we het eventuele branchen van een repository gaan aanpakken bij het opdelen van de implementatie-taken.

### Issues & Milestones

De melding en opvolging van problemen gebeurt ook via GitHub. Dit stelt ons in staat om code-specifieke issues te melden en hiervoor een verantwoordelijke aan te duiden. Via de comments kan er gereageerd worden en kan een probleem worden gesloten wanneer het is opgelost. Dit laatste wordt ook opgevolgd door de Configuration Manager die, buiten de orde en netheid van de repositories te bewaren, ook toezicht houdt over alle open problemen of bugs. Ook voor problemen of opmerkingen bij de documentatie zal hier van gebruik worden gemaakt.

Problemen die geen betrekking hebben tot de code of de documenten, kunnen via de website worden ingediend. De website zal verder ook een overzicht geven (status, beschrijving, ...) van alle ingediende bugs of problemen, zij het of deze werden ingediend via de website zelf of via GitHub.

Aan de hand van de ingediende commits en de milestones wordt de vooruitgang van het project in kaart gebracht. Dit kan worden gebruikt voor evaluaties of het bewerken van het requirements dashboard. Ook deze zullen allemaal in de website worden geïntegreerd.

Om grote problemen of code-verlies tegen te gaan, wordt er enerzijds vertrouwd op de berekenbaarheid en stabiliteit van GitHub, en worden er door de Configuration Manager anderzijds wekelijks back-ups genomen van alle documenten en repositories. Deze back-ups zullen ook via de website ter beschikken worden gesteld.

### 7.1.4 Groei en planning

De in de vorige secties van 7.1 aangehaalde delen, worden gebruikt in functie van het project en zullen bijgevolg (mogelijk) veranderingen of uitbreidingen ondervinden. Het in dit document beschreven SCMP is dan ook een basis voor goed samenwerken en een richtlijn bij de collaboratie. Om flexibele groei toe te staan in correspondentie met het project, stellen we volgende beslissingsprocedures op voor de wijziging van software, implementatie van nieuwe functies of ingebruikname van nieuwe tools.

Het al dan niet adopteren of afkeuren van functies of software, of de manier waarop deze worden gebruikt, zal altijd in samenspraak met het gehele team gebeuren.

De verdere planning en diepere uitwerking van het gebruik, zal verder vorm gegeven worden in de beginfase van de implementatie.

### 7.1.5 SCM Resources

Omdat het gebruikte besturingssysteem afhankelijk is van de gebruiker en kan verschillen per lid, wordt er geopteerd voor software die op zijn minst de laatste versies van de besturingssystemen van Windows en Apple ondersteunt.

Er wordt enkel gebruikt gemaakt van open source software of zelf ontwikkelde tools.

Voor de samenwerkingen maken we, zoals eerder vermeld, gebruikt van onze website, waar we trachten alles te centraliseren, en van GitHub, dat voor elk platform beschikbaar is: meer info op `http://windows.github.com` en `http://mac.github.com` of in sectie 7.1.3. De Configuration Managers, of leden die geavanceerder aan de repositories willen sleutelen, zullen dan ook gebruik maken van SourceTree `http://www.sourcetreeapp.com`.

Code zal worden geschreven in Eclipse[14] en gedocumenteerd in Javadoc[15].

Al deze software zal worden gebruikt volgens de richtlijnen beschreven in dit document en de opdracht.

### 7.1.6 Onderhoud

Alle versies van de documenten zullen worden bijgehouden op GitHub en onder toezicht worden geplaatst van o.a. de Configuration Manager.

Na nieuwe beslissingen te hebben genomen in verband met de configuratie, zal dit worden gereflecteerd in het SCMP dat zo snel mogelijk up to date dient te worden gebracht door de Configuration Manager.

Deze versie wordt gereviseerd door de groep en goedgekeurd alvorens het opnieuw in het SPMP wordt opgenomen.

## 7.2 Verificatie en validatie plan

Hiervoor verwijzen we naar het Software Test Plan (STD) beschikbaar op de website [1].

## 7.3 Documentatie plan

Bij dit project worden de volgende documenten verwacht:

- Software Project Management Plan (SPMP)
  - Software Quality Assurance plan (SQAP als onderdeel van het SPMP)
  - Software Configuration Management Plan (SCMP, ook onderdeel van het SPMP)
- Software Test Plan (STD)
- Software Requirements Specification (SRS)
- Software Design Document (SDD)
- Minutes van alle vergaderingen
- Documentatie bij de source code

De layout van de documenten is vastgelegd volgens de VUB-huisstijl [10], inclusief font-stijl. Het SPMP wort geschreven door de Project Manager, het SQAP en STD door de Quality Assurance Manager, het SRS door de Requirements Manager en het SDD door de Design Manager. Minutes van vergaderingen worden opgesteld door de secretaris. De documentatie van code wordt opgesteld door alle programmeurs en gecontroleerd door de Quality Assurance Manager. Overigens worden zowel source code als alle documenten op kwaliteit gecontroleerd door de Quality Assurance Manager.

De documenten zullen gestockeerd worden op een afzonderlijke repository op GitHub (behalve de documentatie bij de source code). Er zal gebruik worden gemaakt van de webpagina [1] voor het rapporteren en controleren van veranderingen aan de verscheidene documenten. Dit zal gebeuren via de GitHub API [4], hierdoor worden deze wijzigingen ook doorgevoerd op GitHub.

## 7.4 Software Quality Assurance Plan

Op vergaderingen met het team wordt besproken of men nog steeds op schema zit van het voorgestelde ontwikkelingsproces. Wanneer dit niet zo is moet de Project Manager ofwel het team bijsturen, ofwel de verwachtingen veranderen.

### 7.4.1 Documenten en standaard

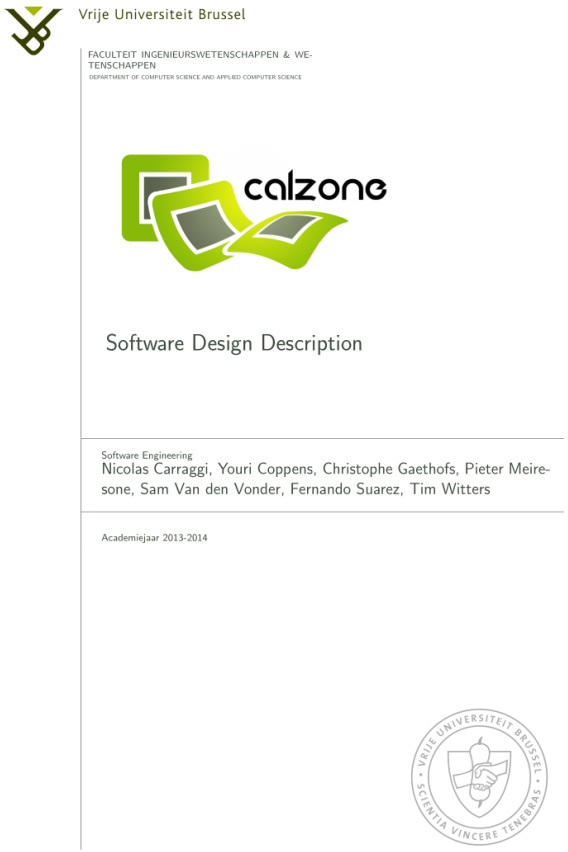
Per iteratie worden 4 (argumenteerbaar 6) documenten opgeleverd die elk door een verschillend persoon worden gemaakt, nl. het SPMP, STD, SRS, SDD, en als onderdeel van het SPMP: het SQAP en SCMP.

Al deze documenten worden gemaakt door hun respectievelijke verantwoordelijke. Daarom is het belangrijk dat de Quality Assurance Manager standaarden vastlegt i.v.m. structuur en opmaak. Alle documenten worden

geschreven in LaTeX en hun structuur, opmaak, spelling en zinsbouw worden gecontroleerd door de Quality Assurance Manager. Als rechtstreeks gevolg zijn er intern deadlines vastgelegd één week voor officiële deadlines. Dit geeft het hele team ruim voldoende tijd om eventuele gebreken op te lossen, maar ook kan de Quality Assurance Manager documenten en source code controleren voor oplevering. Wanneer een document klaar is dient de verantwoordelijke van dit document een mail te sturen (met rechtstreekse URL) via de mailinglijst naar alle leden van de groep. Vervolgens kan de Quality Assurance Manager controleren of het document voldoet aan de afgesproken layout en conventies.

Het is niet de verantwoordelijkheid van de Quality Assurance Manager om documenten te corrigeren, slechts om te controleren en de verantwoordelijke op de hoogte te stellen van eventuele gebreken.

Alle documenten worden opgeleverd in het Nederlands met een aangepaste VUB LateX stijl[10]. Het voorblad van documenten ziet er zo uit:




**Figuur 7.1:** Het standaard voorblad

### 7.4.2 Broncode

#### Documentatie en commentaar

Er wordt verwacht dat de programmeurs hoogkwalitatieve code schrijven, d.w.z. mooie, leesbare code met voldoende commentaar. Meer concreet moet alle code voldoen aan de conventies opgesteld door het voormalige Sun Microsystems[17]. Bovendien moet code voldoende gedocumenteerd worden d.m.v. JavaDoc[15] voor Java,

of equivalent voor andere gebruikte programmeertalen. Deze documentatie moet samen met de commentaar bij broncode voldoende zijn voor de Quality Assurance Manager om code te lezen en begrijpen. Wanneer code niet voldoet aan de vooropgestelde eisen zal de Quality Assurance Manager de programmeur hier onmiddelijk van verwittigen, waarna de programmeur ervoor moet zorgen dat de code wel voldoet aan de eisen. De Quality Assurance Manager zal broncode controleren nadat de programmeur een module heeft afgewerkt.

Uiteindelijk ligt de verantwoordelijkheid voor hoogkwalitatieve code bij de programmeur zelf. De Quality Assurance Manager zorgt er slechts voor dat code voldoet aan een bepaalde standaard, en zo niet moet de programmeur zijn best doen om deze standaard alsnog te halen.

Geschreven code in Java moet voldoen aan de conventies opgelegd door Javadoc [18]. Er wordt extra aandacht besteed door de Quality Assurance Manager aan het formaat van commentaar en de documentatie die hieruit gegenereerd wordt door JavaDoc.

**Unittests**

De Quality Assurance Manager zal er ook op toezien dat modules voorzien zijn van geautomatiseerde tests d.m.v. JUnit voor Java of equivalent voor andere programmeertalen. Niet-triviale methoden en klassen moeten voorzien zijn van Unittests. Wanneer deze tests niet aanwezig zijn voor modules die dit wel vereisen zal de programmeur hierop attent gemaakt worden. Code moet dus voldoen aan de Software Test Documentation (STD).

### 7.4.3 Inspecties

Onderstaande tabel bevat de datum van alle uitgevoerde inspecties, de datum waarop veranderingen aan de code zijn toegepast, alsook een korte beschrijving van het onderdeel dat onderhevig is aan inspectie. Rapporten van de inspectie zijn op GitHub[3] te vinden in een folder "Inspecties". De rapporten van de inspecties zijn gesorteerd op datum wanneer de inspectie is uitgevoerd.

| Inspectie datum | Doorvoering veranderingen | Beschrijving |
|---|---|---|
| 14/11/2013 | 14/11/2013 | Inspectie SPMP |
| 10/12/2013 | Nog niet aangepast | Inspectie SPMP Iter 1 |
| 10/12/2013 | 10/12/2013 | Inspectie STD Iter 1 |
| 11/12/2012 | nog niet aangepast | Inspectie SRS Iter 1 |

## 7.5 Problem resolution plan

Wanneer er inconsistenties gevonden worden in de documenatie of code zullen deze gemeld worden op GitHub m.b.v. issues. Deze issues worden automatisch ook gesynchroniseerd met de website [1]. Deze synchronisatie gebeurt met de GitHub API [4]. Het voordeel van gebruik te maken van de GitHub API is dat alles gecentraliceerd blijf op onze website. Hierdoor is het gemakkelijk om snel een overzicht te krijgen van de huidige stand van zaken.

Vervolgens wordt er een persoon aangewezen die de verantwoordelijkheid krijgt. Voor de documenten is de verantwoordelijke steeds de verantwoordelijke voor het betreffende document (zie hoofdstuk 4). Voor issues in de code wordt de documentatie bij de desbetreffende code geraadgepleegd voor de verantwoordelijke aan te wijzen.

De aangewezen verantwoordelijke is dan verantwoordelijk voor het oplossen van het issue. Dit betekent niet noodzakelijk dat deze persoon het issue effectief moet oplossen. Hij mag ook opdrachten doorgeven aan andere teamleaden op het issue op te lossen. De verantwoordelijke moet er enkel voor zorgen dat het issue opgelost geraakt, ongeacht de gebruikte methode. Vervolgens wordt het issue gesloten door deze verantwoordelijke.

# Bijlage A

# Metrieken

In paragraaf 5.3.6 is er reeds een kort overzicht gegeven van de metrieken die we zullen verzamelen in de loop van dit project. In dit hoofdstuk staat een meer gedetailleerde beschrijving. Op methodenniveau bestuderen we volgende metrieken:

1. Cyclomatic Complexity.

   Deze metriek geeft een indicatie van het aantal 'lineaire' segementen in een methode (m.a.w. stukken code met geen branches). Dit kunnen we onder andere gebruiken om het aantal tests te bepalen om volledige dekking te krijgen. Het geeft ook een indicatie van de complexiteit van de methode.

2. Aantal statements

   Om de grootte van de methode te onderhouden, maken we gebruik van het aantal statements binnen een methode. Deze metriek is robuuster dan het aantal lijnen code vermits deze onafhankelijk is van de gebruikte programmeerstijl.

3. Aantal levels.

   Deze metriek geeft het maximaal aantal geneste lagen in een methode. Een hoog aantal levels geeft aan dat we te maken hebben met een complexe methode. Deze methoden kunnen vereenvoudigd worden door het extraheren van verscheidene private methoden.

4. Aantal lokale variabelen in de scope.

   Deze metriek geeft het maximaal aantal lokale variable dat zich in de scope bevindt gedurende elk mogelijk punt in de methode. Een groot aantal wijst op complexe methoden.

5. Aantal parameters.

   Deze metriek bevat het aantal parameters dat doorgegeven wordt aan een methode. Een te hoog aantal parameters wijst erop dat er te weinig gebruik gemaakt wordt van klassen.

6. Feature envy

   Deze metriek geeft aan in welke mate de methode geïterreseerd is in methoden en attributen van andere klassen. Wanneer deze metriek een hoge waarde heeft, is het beter de methode te verplaatsen naar de klasse waarvan het het meest gebruik maakt. Wanneer dit gedrag slechts deels door een methode wordt bepaald, is het aangewezen om dit deel van de methode te verplaatsen (indien mogelijk).

Voor de berekening gaan we als volgt te werk. Zij $m$ de methode waarvoor we de "feature envy" willen berekenen. We nemen dan $F_c$ de verzameling van features dat gebruikt worden door $m$ in klasse $c$. Verder is $c_m$ de klasse in welke de methode $m$ gedefinieerd is. We definiëren dan de feature envy als:

$$FE = \max_{c \neq c_m}(|F_c|) - |F_{c_m}|$$

Wanneer $FE > 0$ betekent dat de methode $m$ meer features gebruikt van een andere klasse. De klassenstructuur moet dan herzien worden.

Op klassenniveau verzamelen we volgende metrieken:

1. Efferent Couplings.

   In deze metriek wordt er gemeten hoeveel andere klassen de huidige klasse kent. Een hoog aantal koppelingen is nadelig voor de betrouwbaarheid van de code vermits het afhankelijk is van verscheidene types. Door de klasse op te delen in verscheidene deelklassen, kunnen we het aantal koppelingen naar omlaag brengen.

2. Aantal attributen.

   Deze metriek meet het aantal attributen in een klasse. Bij een groot aantal attributen moet er nagegaan worden of er verscheidene attributen kunnen gegroepeerd worden in deelklassen.

3. Complexiteit

   Deze metriek wordt berekend door de som te nemen van de Cyclomatic Complexities van de verschillende methoden die zich in deze klasse bevinden. Deze stelt dus de complexiteit voor van de gehele klasse.

4. Cohesie tussen de verschillende methodes.

   De cohesie tussen de verschillende methoden in één enkele klasse is een belangrijk concept bij object georienteerd programmeren. De cohesie geeft aan of de klasse één of meerdere abstracties voorstelt. Indien één klasse meerdere abstracties voorstelt, moet deze opggesplitst worden in meerdere klassen waarbij elke klassen één abstractie voorstelt.

   Wij zullen de Henderson-Sellers metriek gebruiken [16]. Om deze te berekenen voeren we volgende variabelen in:

   $$M = \{m|\text{m is methode van de klasse}\}$$

   $$F = \{f|\text{f is een veld van de klasse}\}$$

   $$r : F \to \mathbb{N}$$

   Hierbij berekent $r(f)$ het aantal methoden dat veld $f$ aanroept. Vervolgens definiëren we ook $\bar{r}$ als het gemiddelde van $r(f)$ over $F$. De Henderson-Seller metriek wordt dan berekent als:

   $$HS = \frac{\bar{r} - |M|}{1 - |M|}$$

   Hoe lager de waarde, hoe beter de cohese tussen de verschillende methoden.