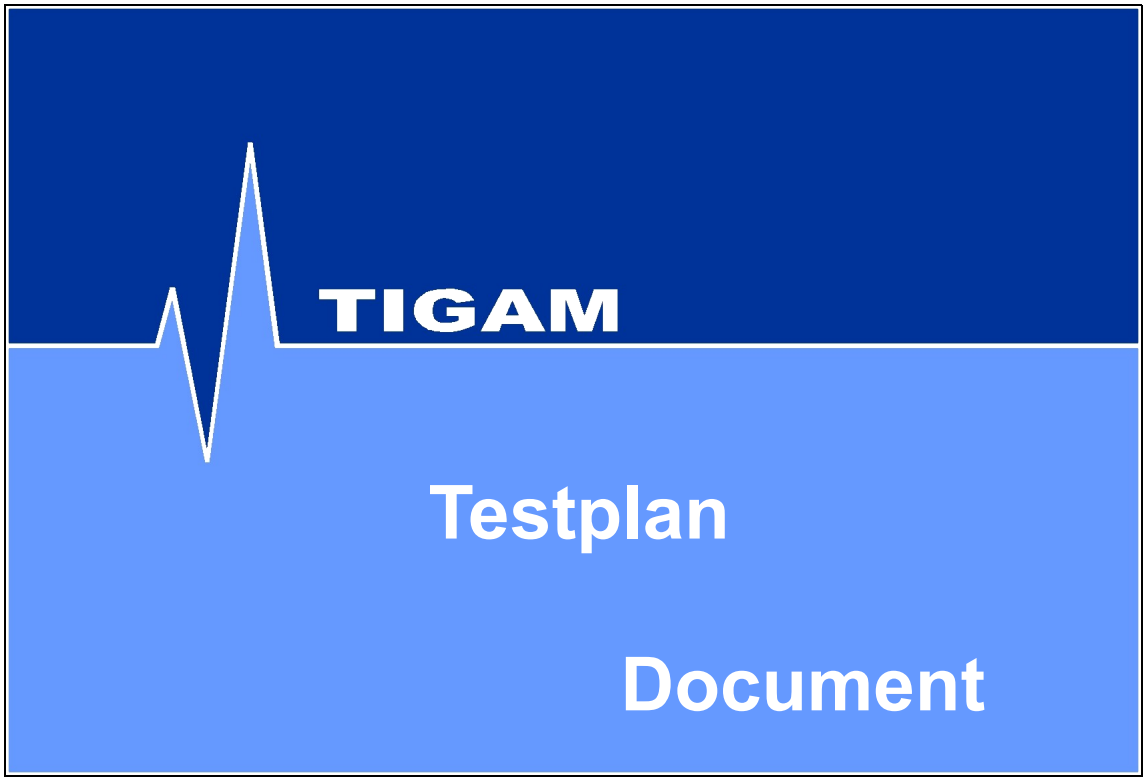

Auteur(s): Mustapha Bouzaidi
Sherwin Angelo
Kevin Wong
Isabelle Hanraads
Versie: 1.1
Status: Ter Inzage
Datum: 17 december 2007

TIGAM is onderdeel van de Hogeschool van Amsterdam

Samenvatting

Dit document bevat de Testplan voor een ontmoetingsportaal. Dit document beschrijft de manier van testen en volgt de IEEE-standaard voor testplan documenten. Dit project is een onderdeel van het vak Software Engineering gegeven aan de Hogeschool van Amsterdam door Dhr. Derriks

Versiebeheer:

Ver.	Status	Datum	Auteur(s)	Wijzigingen
1.0	concept	17 december 2007	Team 1	Document

Inhoudsopgave

1 Introductie

1.1 Doel en bereik van dit document

Dit document beschrijft de manier van testen voor het project van team 1 van Railcab, Hogeschool van Amsterdam. Het project, Railcab genaamd. Deze Testplan beschrijft een manier om de code te testen via specifieke gemaakte Java-klassen. Zie PID voor gedetailleerde specificaties van dit project.

1.2 Definities, acroniemen en afkortingen

PID: Project Initiatie Document

1.3 Referenties

De referenties bevinden zich op het einde van het document.

2 Test plan

2.2 Introductie

Het test plan specificeert de eisen omtrent het testen van de code en van de applicatie gemaakt door team 1. Het beschrijft welke en hoe items getest moeten worden. Enkele van de specificaties zijn de volgende:

- Hoe gaat er getest worden?
- Hoe worden de test georganiseerd?
- Wat moet er getest worden?
- Wat zijn de benodigdheden om te testen?

2.3 Test items

De verschillende documenten beschrijven de items die getest moeten worden:

- De designspecificaties zoals voorzien in het Designmodel
- De kwaliteitsvereisten zoals voorzien in het PID
- De functionele en niet-functionele eisen zoals voorzien in het analysemodel

2.4 Features die getest worden

Alle features worden getest.

2.5 Features die niet getest worden

Geen.

2.6 Aanpak

2.6.1 Component test

Elk component van de software dat gemaakt wordt door een programmeur, moet door dezelfde programmeur getest worden. Men gaat uit van het black-box principe. De code mag geen errors bevatten, de code moet voldoen aan de coding conventions en moet consistent zijn met het design.

2.6.2 Integratie test

Zodra een bepaald software component de component test heeft doorstaan en goedgekeurd is, kan deze gelinkt worden aan andere, al bestaande, componenten. Deze groep van samenwerkende componenten moet getest worden door de programmeur die deze linking doet. De integratie test is geslaagd als de gelinkte componenten zonder errors kunnen samenwerken.

2.6.3 Interface test

Bij het maken van software waar de graphical user interface (GUI) belangrijk is, is het ook nodig om deze uitvoerig te testen. Hier moet men dan eerder kijken naar de usability requirements. De GUI moet gebruiksvriendelijk zijn en alle functionaliteit van de software op een aangename manier aanbieden aan de klant.

2.6.4 Regressie test

De software moet in zijn geheel getest worden.

2.6.5 Veiligheidstest

De software zal getest worden op zijn robuustheid tegen incorrecte invoer en onvoorziene bugs of errors.

2.6.6 Stabiliteitstest

De manier waarop de software reageert tegenover failures, bijvoorbeeld de connectie naar de server die verloren gaat, wordt in deze test bestudeert.

2.6.7 Performantietest

Het testen van de performantie van het systeem houdt in dat de eisen van de klant nageleefd worden, zoals voorzien in het PID.

2.7 Goedkeur- en afkeurcriteria

Een test is geslaagd wanneer alle test items succesvol getest zijn. Voor een component test houdt dit in:

- Voldoet aan functionele en/of niet-functionele eisen.
- Voldoet aan de design-specificaties.
- Voldoet aan de kwaliteitsvereisten.
- Voldoet aan de coding conventions.
- Errorvrij

Wanneer aan deze eisen voldaan zijn, kan het component gebruikt worden in het systeem.

2.8 Uitstellings- en hervattingscriteria

Wanneer een component een test faalt, moet de programmeur de groep op de hoogte brengen zodat een oplossing voor het probleem kan gevonden worden.

2.9 Test deliverables

Een rapport moet ingediend worden voor elke test.

2.10 Omgevingsbenodigdheden

De benodigdheden zijn dezelfde als die voor het gehele project.

2.11 Verantwoordelijkheden

Elke programmeur is verantwoordelijk voor het testen van zijn eigen componenten. Wil de programmeur deze componenten integreren met andere, dan dient hij ook een integratie test uit te voeren.

3 Test design specificatie

3.1 Doel

Deze sectie beschrijft op welke manier de tests zullen gebeuren.

3.2 Aanpak

3.2.1 Introductie

SE2 zal zijn eigen vorm van tests gebruiken. Hiervoor worden geen externe libraries van Java gebruikt. Er werden eerst tests met JUnit uitgevoerd, maar nadien werd er beslist om een eigen test-systeem uit te bouwen.

3.2.2 Try-klasses

Een Try-klasse is een klasse met een main() erin die alle methods in zijn overeenkomstige klasse zal testen.

3.2.3 Try klasses implementeren

Elke method van een klasse moet getest worden in een Try-klasse. Er wordt getest of voor een bepaalde input, de output wel de gewenste output is of niet. De programmeur die een Try-klasse maakt, zorgt ervoor dat deze werkt en dat de tests positief zijn. Dit proces wordt herhaald voor elke klasse uit het project.
Normale structuur

```
public class TrySomeClass {
public static void main(String[] args) {
(hier worden tests uitgevoerd op SomeClass)
}
}
```

4 Test logs

4.1 Doel

Elk lid van de groep zal de logs van zijn tests bijhouden.

4.2 Logs van gebeurtenissen en activiteiten

Wanneer een test compleet is, houdt de verantwoordelijke voor die test de logs bij. Zoals normaal wordt alleen de output van de test in dit log bestand bijgehouden. Via de Try-klasses zijn dit gewone printlines naar de System.out van Java.

4.3 Onvoorziene gebeurtenissen

Wanneer een test faalt en de persoon in kwestie het probleem niet kan oplossen, wordt er een log van deze test bijgehouden. Het probleem zelf wordt dan op bugzilla toegevoegd.