Vrije Universiteit Brussel

FACULTEIT INGENIEURSWETENSCHAPPEN & WETENSCHAPPEN
DEPARTMENT OF COMPUTER SCIENCE AND APPLIED COMPUTER SCIENCE

Software Test Documentation

Software Engineering

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

Academiejaar 2013-2014

Versiegeschiedenis

Tabel 1: Versiegeschiedenis

Versie	Datum	Auteur(s)	Commentaar
1.0	30/11/2013	Sam Van den Vonder	Initiële versie
1.1	10/12/2012	Sam Van den Vonder	Aanpassingen kwaliteitsinspectie
2.0	19/02/2014	Sam Van den Vonder	Meer gedetailleerde informatie testing
2.1	27/02/2014	Sam Van den Vonder	Gedetailleerdere informatie m.b.t. code conventies

Inhoudsopgave

Lijst van tabellen

Hoofdstuk 1

Introductie

In dit document wordt de strategie besproken voor het schrijven van tests in de CalZone [**?**] applicatie. Door het testen van geschreven code kan de werking en functionaliteit van de applicatie worden gegarandeerd. Deze werking en functionaliteit staat beschreven in het SRS [1] en het SDD [2]. De Quality Assurance Manager is verantwoordelijk voor dit document, en zorgt dat dit plan opgevolgd wordt.

Het correct opvolgen van dit document bespaart de programmeur niet alleen veel moeite en hoofdpijn, maar kan de werking van de applicatie en zijn features garanderen.

Hoofdstuk 2

Referenties

[1] *CalZone* http://wilma.vub.ac.be/~se2_1314

[2] *Software Requirement Specification document* https://github.com/CalZoneVUB/Documents/tree/master/SRS

[3] *Software Design Document* https://github.com/CalZoneVUB/Documents/tree/master/SDD

[4] *Software Project Management Plan* https://github.com/CalZoneVUB/Documents/tree/master/SPMP

[5] *IEEE Std 829-2008™ , IEEE Standard for Software and System Test Documentation* http://ieeexplore.ieee.org/servlet/opac?punumber=4578271

[6] *JUnit* http://junit.org/

Hoofdstuk 3

Definities

Tabel 3.1: Overzicht van de gebruikte acroniemen.

Acroniem	Betekenis
STD	Software Test Documentation
SRS	Software Requirement Specification
SDD	Software Design Document
SPMP	Software Project Management Plan
GUI	Graphical User Interface

3.1 Overzicht

Dit document volgt de IEEE standaard voor het opstellen van een systeem test document [4]. Methoden die gebruikt worden bij het testen van de applicatie worden in dit document besproken.

3.1.1 Scope

Dit document is onderdeel van het CalZone[**?**] project. Met behulp van dit project kunnen lessenroosters aangemaakt en bekeken worden op zowel traditionele computers als smartphones. De volledige beschrijving van het project is terug te vinden in de bijhorende documenten: het SDD[2], SPMP[3] en de SRS[1].

Hoofdstuk 4

Software Test Plan

4.1 Inleiding

Het systeem kan op verschilende manieren getest worden. Langs de ene kant zijn er tests voor onafhankelijke stukken code (Unittests), en langs de andere kant wordt de samenwerking van deze componenten getest (Integratietests). Daarnaast moet op vlak van GUI getest worden of de applicatie stabiel blijft in bepaalde scenario's, of bij foute input. Door middel van tests moet gegarandeerd worden dat het systeem werkt en voldoet aan de specificaties beschreven in het SRS[1]. Wanneer de programmeur code wenst te pushen naar de hoofdrepository moet deze eerst alle tests successvol uitvoeren. Wanneer voldoende tests zijn geschreven garandeert dit dat het systeem werkt zoals verwacht.

4.2 Soorten tests

4.2.1 Unittests

Unittests zijn tests die kleine, onafhankelijke stukken code testen. Code die door unittests getest wordt mag geen gebruik maken van externe bronnen, anders kan men niet nagaan of de fout in de te testen klasse/methode zit, of in de externe code. Wanneer een klasse wordt afgewerkt (of een subset van zijn methoden) moet hiervoor reeks unittests worden geschreven. Deze tests moeten correctheid van een individueel component nagaan. Wanneer de klasse wordt aangepast kunnen deze tests opnieuw gerunt worden. Hierdoor kunnen mogelijke problemen vroeg gedetecteerd worden. Deze tests bevatten o.a. ook tests over de database, die vaak als een aparte categorie worden beschouwd.

4.2.2 Integratietests

Bij unittests werd vermeld dat de klassen geen gebruik mogen maken van externe bronnen. Wanneer dit wel zo is, en dus de samenwerking tussen de twee componenten wordt getest, zijn dit integratietests, m.a.w. hoe twee of meer componenten met elkaar integreren. Gewoonlijk vangen integratietests meer bugs en regressies dan unittests, daarom moet de programmeur extra aandacht besteden aan het schrijven van dit soort tests.

4.2.3 Verificatietests

Verificatietests testen en garanderen dat het systeem voldoet aan de requirements beschreven in het SRS[1]. Deze tests zijn niet mutueel exclusief met unittests en integratietests, maar zijn in sommige gevallen wel expliciet nodig.

4.2.4 GUI tests

De GUI wordt ook uitvoerig getest. Deze moet blijven werken zoals verwacht met arbitraire input. Deze tests worden eveneens geautomatiseerd.

4.3 Tools

Door middel van JUnit 4[5] worden tests opgesteld voor het project. JUnit is een standaard Java library en overigens geïntegreerd in de user interface van Eclipse, waardoor het uitvoeren van tests makkelijk wordt en zodat men makkelijk conclusies kan trekken nadat alle tests gerunt zijn. Met JUnit kunnen makkelijk tests geschreven worden d.m.v. "assertions". Een assertion test of een echte waarde (verkregen door het uitvoeren van een methode) gelijk is aan de verwachte waarde opgegeven door de programmeur. Wanneer dit het geval is faalt de test.

4.4 Criteria voor succesvolle tests

Wanneer er geen fouten gevonden worden in een klasse d.m.v. de bijhorende tests zeggen we dat deze slaagt. Indien er zich wel fouten voordoen moeten deze worden opgelost alvorens de code mag worden opgenomen in de hoofdrepository. De programmeur die een fout omerkt maakt steeds een issue aan op GitHub indien hij zelf niet verantwoordelijk is voor de falende code. Deze bug reports worden toegewezen aan de verantwoordelijke voor de falende code, welke de code zo snel mogelijk moet verbeteren. Wanneer een test faalt in onverwachte omstandigheden, m.a.w. in code die de programmeur niet heeft aangepast maar eventueel wel gebruikt, moet hiervoor ook een issue geplaatst worden op GitHub.
Deze issue bevat de volgende informatie:

- Het requirement ID, zoals gespecifieerd in het SRS[1]
- Een beschrijving van het probleem
- Input die het probleem veroorzaakt
- De verwachte output
- De daadwerkelijke output

Deze issue kan daarna opgelost worden door de programmeur verantwoordelijk voor de falende code, of door bijdrage van andere programmeurs.

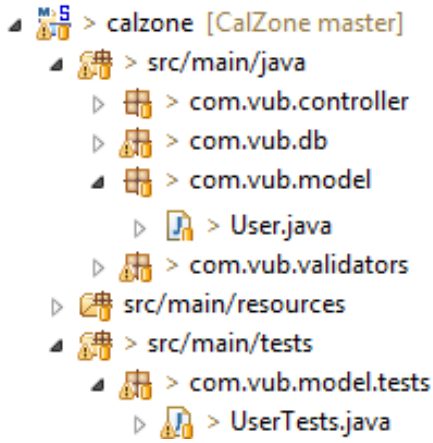
4.5 Verantwoordelijkheden

De programmeur is verantwoordelijk voor het schrijven van tests voor zijn afgewerkte onderdelen. Code waar geen tests voor geschreven zijn mag niet worden opgenomen in de hoofdrepository op GitHub. Zonder tests kan de werking van deze code eenmaal niet worden bevestigd. De Software Quality Assurance Manager is verantwoordelijk voor het controleren dat de programmeurs deze tests volledig en correct schrijven.

4.6 Indeling

Alle tests worden bijgehouden in een aparte source folder in het project, genaamd *src/main/tests*. De directorystructuur in deze folder is dezelfde als die van de source code van het project. Bronbestanden zijn georganiseerd in packages, en dat is met tests exact hetzelfde, maar met een *.tests* suffix. De java-klasse bronbestanden krijgen het *Tests* suffix.

Hier is één groep tests beschikbaar, nl. de tests die horen bij het bronbestand *User.java* in het package *com.vub.model*. De eigenlijke tests op dit bestand staan in de directory *src/main/tests*. De bijhorende package noemt *com.vub.model.tests*, met bestandsnaam *UserTests.java* voor de tests.



Figuur 4.1: Directorystructuur van testbestanden.

4.7 Belangrijkste Java code conventies

4.7.1 Naming conventions

Klassen & interfaces

Gebruik simpele, maar beschrijvende namen (zelfstandige naamwoorden) die iets zeggen over de klasse. Elk zelfstandig woord in de klassenaam begint met een hoofdletter. Vermijd acroniemen, tenzij deze algemeen gebruikt worden (zoals URL en HTTP).

class Course;
class ProfilePageController;

Methoden

Namen van methoden beginnen met werkwoord (en een lowercase letter), met hoofdletters voor elk volgend woord.

run();
runTests();
runTestInBackground();

Variabelen

Variabelen beginnen met een lowercase letter, met opeenvolgende woorden gestart worden met een hoofdletter. De naam van een variabele moet kort, maar toch beschrijvend zijn. De namen van constante variabelen in een klasse zijn volledig uppercase, met opeenvolgende woorden gescheiden door een underscore.

float width;
float minWidth;
int MIN_WIDTH = 1;

boolean DEBUG = false;

Bovendien wordt ook één declaratie per lijn aangemoedigd, omdat dit het schrijven van commentaar kan aanmoedigen.

float width, height; *// fout*
float width; *// Width of the bridge*
float height; *// Height of the bridge*

4.7.2 Klasse en interface declaraties

- Bij aanroep van een methode, geen spatie tussen de methode naam en het openende haakje
- De open brace {staat op dezelfde lijn als de naam van de declaratie
- De sluitende brace staat automatisch op een nieuwe regel, tenzij de methode een lege body heeft
- Methoden zijn gescheiden door een blanke regel

Voor elke klasse staat een comment met het volgende template:

```
/*
* Klasse naam
*
* Versie info
*
* Auteur
*
* Beschrijving van de klasse
*/
```