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
3	11/04/2014	Sam Van den Vonder	Revisie en aanvulling document
3.1	17/04/2012	Sam Van den Vonder	Toevoegen huidige tests

Inhoudsopgave

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 van gebruikers. Door middel van tests moet gegarandeerd worden dat het systeem werkt en voldoet aan de specificaties beschreven in het SRS[1]. Programmeurs mogen slechts code pushen naar de hoofdrepository wanneer alle aanwezige tests nog steeds correct uitgevoerd worden. Wanneer voldoende tests zijn geschreven voor alle relevante onderdelen garandeert dit dat het systeem werkt zoals men 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. Bovendien kan Unittests testen ook het correct opslaan van data in de database, hoewel dit vaak als een aparte categorie wordt 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. Integratietests testen dus de samenwerking tussen verschillende componenten, in plaats van de werking van slechts 1 component. 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 kunnen omvat worden in unittests en integratietests, maar in sommige gevallen is verificatie van bepaalde functionaliteit expliciet nodig.

4.2.4 GUI tests

De GUI wordt ook uitvoerig getest. Deze moet blijven werken zoals verwacht met arbitraire input. De automatisering van deze tests wordt vaak uitgevoerd door middel van software zoals Selenium, maar vaak is dit niet nodig vanwege het onderliggende framework waarmee de webapplicatie is opgebouwd.

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. Hierdoor wordt het uitvoeren van tests makkelijk, en kan men men makkelijk conclusies kan trekken uit de resultaten die voortvloeien uit de tests. Met JUnit kunnen makkelijk tests geschreven worden d.m.v. "assertions". Een assertion test of een verkregen waarde (door het uitvoeren van een methode) gelijk is aan de verwachte waarde opgegeven door de programmeur. Wanneer deze waarden verschillen faalt deze specifieke test. In de applicatie worden uiteraard veel tests geschreven, waardoor al deze tests tezamen de werking van het systeem garanderen.

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 opmerkt 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:

- 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 van het project

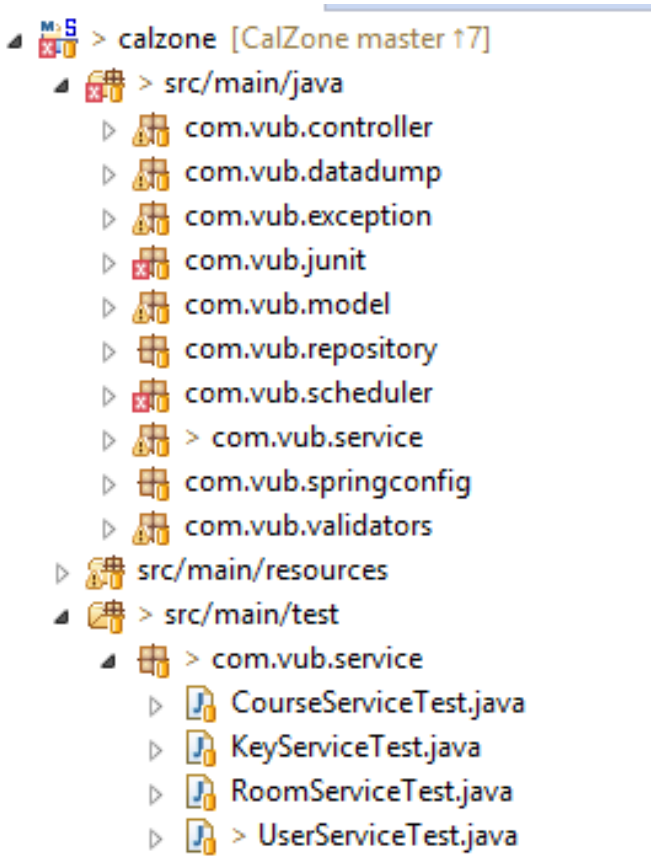
Alle tests worden bijgehouden in een aparte source folder in het project, genaamd *src/main/tests*. De directorystructuur in deze folder is dezelfde als die van de source code van het project. De source code van het project is in packages, en dat is met tests exact hetzelfde. De bronbestanden voor tests krijgen dezelfde naam als het bestand waar deze tests over gaan, maar zij krijgen ook het *Test* suffix.
Het project is op deze manier ingedeeld voor source folder *src/main/java*

- **com.vub.controller**: Hierin zitten alle Spring controllers, m.a.w. de klassen die alle requests van de front-end webpagina's verwerken.
- **com.vub.datadump**: Bevat alle klassen die nodig zijn om de gegeven datadump in te laden in de database.
- **com.vub.exception**: Bevat alle custom exceptions die gethrowed worden in de code. In plaats van *null* values terug te geven uit methoden geven wordt de voorkeur gegeven aan een exception, vermits een exception nooit ambigu kan zijn. De betekenis van een *null*-waarde kan het gevolg kan zijn van veel factoren, en kan dus ook verschillend geïnterpreteerd worden.
- **com.vub.model**: In het *model-view-controller* patroon bevat het model alle data van de applicatie. Vermits de modellen rechtstreeks op de database worden gemapt kan gesteld worden dat deze modellen de database 'in geheugen' bevatten. In de modellen zit geen business logic, d.w.z. dat hierin geen methoden staan die ingewikkelder zijn dat standaard getters en setters.
- **com.vub.repository**: Zoals hierboven vermeld worden alle modellen rechtstreeks op de database gemapt. Om de link te vormen tussen de database en de modellen is een *repository* nodig. Deze repository biedt standaard methoden voor alle CRUD operaties en een beperkte selectie methoden voor het zoeken van data in de database.
- **com.vub.scheduler**: Deze package bevat alles m.b.t. de scheduler
- **com.vub.service**: De modellen bevatten geen business logic, en de repositories verzorgen alleen de directe link naar de database. De services vormen een abstractielaag boven de repositories en bevatten alle business logic m.b.t. de modellen. Dit zijn de klassen waarmee ontwikkelaars rechtstreeks mee in contact komen, terwijl de repositories voor hen volledig verborgen zijn.
- **com.vub.springconfig**: Deze package bevat eventuele klassen die aangemaakt zijn om Spring te configureren.
- **com.vub.validators**: Op gebied van validatie in de user interface kunnen *validators* aangemaakt worden. Hiermee gaat input van gebruikers geldig of ongeldig verklaard worden.

4.7 Testing van onderdelen

Niet alle onderdelen zijn even belangrijk om te testen. Neem bijvoorbeeld de modellen in *com.vub.model*, deze bevatten alleen getters/setters, en zijn daarom oninteressant om tests op uit te voeren. De meest belangrijke onderdelen om te testen zijn de volgende:

- **com.vub.scheduler**: De scheduler houdt zich aan bepaalde regels waaraan een planning moet voldoen. Om te garanderen dat deze regels (constraints) correct werken kunnen deze onderworpen worden aan unittests.
- **com.vub.service**: De service-laag kan complexe operaties bevatten die werken op de modellen, dus deze operaties moeten uitvoerig getest worden. Bovendien moet ook getest worden of de modellen en alle bijhorende relaties correct opgeslagen worden in de database. Dit hoort ook bij het testen van services.
- **com.vub.validators**: Input van gebruikers in de user-interface wordt gevalideerd d.m.v. deze validators, dus daarom is het belangrijk dat deze correct werken. Het is niet nodig om de user-interface te testen d.m.v. een ander framework of softwarepakket, vermits alle user-input gevalideerd kan worden d.m.v. validators. Daarom moet ervoor gezorgd worden dat deze validators correct werken.



Figuur 4.1: Directorystructuur van testbestanden.

4.8 Uitgevoerde tests

In de applicatie worden verschillende tests uitgevoerd. In onderstaande tabel staan alle uitgevoerde tests. Testklasse duidt de klasse aan (.java bestand) in het project waarin de test wordt uitgevoerd, en testnaam is de naam van de test. In een testklasse kunnen meerdere tests bestaan. Onderdeel slaat op het onderdeel van de applicatie waar de test zich mee bezig houdt - dit wil zeggen opslag van data in de database, testen van regels in de scheduler, enz...

Tabel 4.1: Tests in de applicatie

Testklasse	**Testnaam**	**Onderdeel**	**Beschrijving**
CourseServiceTest	testCourseCreation	Database	Test het aanmaken van een Course in de database (correcte opslag van velden)
KeyServicetest	testKeyCreation	Database	Tests m.b.t. aanmaken van keys in de database
RoomServiceTest	testRoomCreation	Database	Test of een lokaal kan aangemaakt worden in de database
UserServiceTest	testUserCreation	Database	Test of een user kan aangemaakt worden in de database
UserServiceTest	testUserActivation	Activatie	Test of users succesvol geactiveerd kunnen worden (alleen met een activatie-key)
EmailValidatorTest	notRealEmail	Validatie	Test of foute e-mailadressen niet geaccepteerd worden
EmailValidatorTest	notRealEmail2	Validatie	Andere test op foute-email adressen
EmailValidatorTest	realEmail	Validatie	Test of een echt e-mail wordt geaccepteerd
SchedularTest	overlappingTeacherAgenda-Explicit	Schedular	Test of 4 vakken gescheduled kunnen worden zonder overlap (met dezelfde prof, in één lokaal)
SchedularTest	overlappingTeacherAgenda-Implicit	Schedular	Test of 4 vakken gescheduled kunnen worden zonder overlap (in één lokaal)
SchedularTest	simpleSchedulingWith-Teachers	Schedular	4 vakken moeten gescheduled worden in 2 tijdslots met maar 2 lokalen
SchedularTest	schedulingRangeTest	Schedular	4 vakken moeten gescheduled worden in veel tijdslots met één lokaal
SchedularTest	roomAllocationByCapacity	Schedular	Scheduling van vakken, rekeninghoudend met capaciteit van lokalen
SchedularTest	preventAdjacentLectures-OfSameCourseCompo-nent	Schedular	Zorg dat HOC en WPO voor één vak niet overlapt
SchedularTest	noonBreak	Schedular	Zorg dat er altijd een uur middagpauze is
SchedularTest	correctRoomType	Schedular	Test of een vak altijd in een correct lokaal gepland wordt (leslokaal/computer lokaal)
SchedularTest	advancedScheduling	Schedular	Probeer een volledige week te schedulen

4.9 Belangrijkste Java code conventies

Ook bij het testen van de applicatie moeten de Java conventies gehanteerd worden. Hieronder staan de belangrijkste.

4.9.1 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. Merk ook op dat in het geval van booleaanse waarden, de getters en setters steeds beginnen met is, bijvoorbeeld *isProjectorEquipped* (m.b.t. projectors in een lokaal). Dit in tegenstelling to het foute *hasProjector*.

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*

4.9.2 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