Software Test Plan

Yannick Verschueren

April 2015

Document geschiedenis

Versie	Datum	Auteur/co-auteur	Beschrijving
1	November 2014	Yannick Verschueren	Eerste versie
2	December 2014	Yannick Verschueren	Tweede versie
3	Maart 2015	Yannick Verschueren	Derde versie
4	April 2015	Yannick Verschueren	Vierde versie

Inhoudstafel

1 Introductie

1.1 Domein

Het Software Test Plan behandelt alle methoden, gebruiken en standaarden die van toepassing zijn bij het testen van het "WiseLib" project. Het domein van de testen beschreven in dit plan beschouwt:

1. Het testen van de functionele vereisten,beschreven in het SRS
2. Controleren van de gebruikers vereisten
3. Verzekering van code kwaliteit
4. Unit testen en verwijderen van bugs

Het verschil tussen functionele verereisten en gebruikers vereisten wordt duidelijker met een voorbeeld. Een van de functionele vereisten is het verwijderen van een publicatie uit de persoonlijke lijst van een gebruiker. Deze vereiste zegt echter niet op welke manier dit moet gebeuren. Een specifieke gebruikersvereiste zegt dan hoe dit precies moet gebeuren. Moet de gebruiker op een knop naast de publicatie drukken? Moet er een speciaal menu gebruikt worden?

1.2 Doel

Het doel van dit document is om een overzicht te geven van alle strategieën, procedures, activiteiten en taken die instaan voor het testen van het project. Het document volgt de IEEE standaard voor Software en Systeem Test Documentatie.[1]

1.3 Objectieven

1.3.1 Primair objectief

Software projecten bestaan uit verschillende onderdelen: software, hardware en interfaces. De testing procedures verder beschreven in dit document behandelen elk onderdeel met variërende mate. Zo zal het testen van de hardware miniem of onbestaand zijn, aangezien de server waarop de applicatie draait, niet in de handen van de ontwikkelaars is. Software en interfaces worden wel getest worden, alhoewel de focus op de software ligt. Variërende mate: hardware wordt niet zawaar getest, interface wordt meer getest, software wordt het meeste getest.

Software De geïmplementeerde software is de basis van het project en wordt dus met bijhorende intensiteit getest. Hierdoor wordt een aanzienlijk deel van de tijd gespendeerd aan het testen van de code.

Hardware Het systeem draait op twee verschillende types hardware. Het cliënt gedeelte werkt op het apparaat van de gebruiker en is dus buiten het bereik van de tester van het systeem. De server waarop het systeem draait is nogmaals niet in het bezit van het team en is dus geen verantwoordelijkheid voor de tester.

[1] `http://standards.ieee.org/findstds/standard/829-2008.html`

Interface De functionaliteit van de interface wordt door zowel test manager als webmaster getest.

De test procedures zorgen ervoor dat in het systeem:

1. De applicatie op een correcte manier functioneert, dit houdt in dat de applicatie zonder fouten en bugs kan worden gebruikt.
2. De functionele vereisten vervult zijn.
3. De gebruikers het systeem correct en zoals beoogd kunnen gebruiken. Met andere woorden, de gebruikers vereisten vervuld zijn. Dit houdt in dat de voorgeschreven scenario's op een correcte manier door de applicatie worden afgehandeld.

1.3.2 Secundair objectief

Het secundair objectief in het testen van een systeem is het opsporen en behandelen van alle problemen en risico's, deze problemen delen met het team achter het project en het oplossen van deze problemen op een correcte manier.
Hierbij ligt de nadruk op het efficiënt en correct communiceren van problemen tussen leden van het team. Dit document beschrijft alle tools en platformen die gebruikt worden om problemen te melden en op te lossen.

2 Referenties

- Software Configuration Managment Plan
- Software Design Document

3 Definities, acroniemen en afkortingen

3.1 Acroniemen

SPMP Software Project Management Plan

SRS System Requirement Specifiaction

STD Software Test Document

SDD Software Design Document

3.2 Definities

Integriteit niveau is een indicatie van de relatieve belangrijkheid van een software onderdeel tot de stakeholders of opdrachtgevers. Niet te verwarren met itensiteit ; integriteit hoort bij een functionaliteit, terwijl intensiteit bij de test procedures, die de functionaliteit controleren, hoort.

Unit test methode om software modulen of stukken broncode (units) afzonderlijk te testen.

Integratie test fase waarin individuele software modules gecombineerd worden en getest worden als een groep.

4 Integriteit niveau's

Elk onderdeel van het systeem heeft een bepaald integriteit niveau (zie definities in sectie 3). In geval van het falen van een onderdeel zegt de integriteit in hoeverre dit het gehele systeem zal beïnvloeden en hoe belangrijk het oplossen van het probleem is. Hoe hoger het niveau, hoe meer test zullen worden uitgevoerd. Er bestaan drie niveaus:

1. De functionele vereisten.(Hoge prioriteit)
2. De scenario's of use-cases. (Gemiddelde prioriteit)
3. Aanvullende functionaliteiten. (Lage prioriteit)

De functionele vereisten hebben de hoogste graad van belangrijkheid aangezien zij opgelegd zijn door de opdrachtgevers. Deze functionaliteiten moeten aanwezig zijn in de applicatie en moeten feilloos werken. De scenario's zijn opgelegd door de ontwikkelaars en zijn afgeleid uit de functionele vereisten. Deze scenario's bepalen op welke manier een functionele vereiste zal worden geïmplementeerd en hoe de gebruiker een functionaliteit zal gebruiken. De ontwikkelaars bespreken wat concreet moet bereikt worden door een functionele vereiste en beslissen dan wat de optimale implementatie en gebruikersomgeving zal zijn die functionaliteit ondersteund. Deze vereisten hebben een lager niveau aangezien ze niet noodzakelijk zijn voor de opdrachtgevers. Het is belangerijker dat een functionaliteit volledig werkt, dan dat ze gebrekkig werken maar voldoen voor de scenarios. De opdrachtgevers kunnen wel richtlijnen geven indien de interface niet aan hun verwachtingen voldoet. De aanvullende functionaliteiten hebben de laagste graad aangezien zij weinig of geen belang hebben in het correct functioneren van de applicatie.

5 Test processen

Het Mocha [2] framework wordt gebruikt in combinatie met Should.js [3] als algemeen test platform op de server.
Het testen van de code die op het apparaat van de gebruiker wordt gedaan aan de hand van Karma [4] in combinatie met Mocha. De integratie van Karma met Mocha zorgt ervoor dat het schrijven van client tests en server tests geen verschillende kennis vereist.

Mocha wordt gebruikt omdat het testplatform gebruik maakt van node.js. Onze applicatie werkt ook met node.js, dus het was een logische stap om dit platfom te gebruiken in onze test processen. Daarnaast is bevat Mocha een groot aantal features, waaronder het eenvoudig testen van asynchrone procedures. Omdat onze applicatie aanzienlijk gebruikt maakt van een database, bevat de applicatie veel asynchrone code en is er dus nood aan een mogelijkheid om deze code te testen. Should.js is een van de populairste testing framework library. Het bevat een gemakkelijke en leesbare syntax waarin de testen kunnen geschreven worden.

[2] `http://mochajs.org/`
[3] `https://github.com/tj/should.js`
[4] `http://karma-runner.github.io/`

Het Mocha framework maakt gebruik van de library om de staat van een object te controleren. Bij het starten van mocha wordt een node.js process gestart, worden alle modules geladen en wordt alle code uitgevoerd die opgeroepen wordt in de tests. De library wordt dan gebruikt om na te gaan of een object de juiste attributen bevat en of er geen fouten onstaan bij het uitvoren van de code. Een speciale caalback in de tests kan gebruikt worden bij het testen van asynchrone code. De test zal wachten tot de callback is uitgevoerd, voordat het framework de volgende test uitvoert. De library, in dit geval Should.js, voegt aan elk object een speciale method ('should') toe. Deze method laat toe om attributen in een object te vergelijken met een waarde en een fout terug te geven aan het testing framework indien nodig. Karma is een testing framework geschreven voor angular.js. De keuze van dit framework is opnieuw logisch aangezien de client zijde van onze applicatie ook met agular.js werkt. Ook de mogelijkheid tot integratie met Mocha was een reden om te kiezen voor Karma.
De intensiteitsgraad en strengheid in het uitvoeren en documenteren van test procedures is evenredig met het integriteit niveau van de vereiste. Hoe lager het niveau, hoe lager de intensiteit en strengheid van de test procedures. Hoe hoger de het niveau, hoe belangerijker de functionaliteit is en hoe strenger de test procedures zijn.

De test methodologie wordt besproken in het Quality Assurance onderdeel van het SPMP.

5.1 Unit testing, integratie testing en validatie testen

Alle integratie tests worden ontworpen en uitgevoerd door de Test Manager. De integratietest testen de werking van verschillende samenwerkende modules. Unit tests worden onderhouden door de test manager, maar worden geschreven en uitgevoerd door de ontwikkelaars van de bijhorende functionaliteit. De test-manager zorgt ervoor dat alle unit tests correct zijn en zal ook zelf indien nodig de test schrijven, indien het niet in het tijdschema van de ontwikkelaar valt.

5.1.1 White-box tests

Zowel de unit tests als de integratie tests zijn white-box tests, integratie tests worden geïplementeerd met kennis van het gehele systeem. Zij testen de interne structuren en werking van het programma. Unit tests daarentegen kennen ook de interne werking, maar enkel van de functionaliteit waarvoor zij geschreven zijn. Het gemeenschappelijke element is hier de kennis van de implementatie van de functionaliteit, wat rechtegenover de black-box tests staat.

De Should.js biblitotheek wordt gebruikt bij de unit en integratie tests van de geschreven code.
Het Mocha framework zorgt ervoor dat alle tests op een correcte manier worden uitgevoerd. Een simpel commando voert alle test uit die zich in de "tests" map bevinden. Deze map wordt opgesplitst in verschillende modules om een overzicht te houden over de verschillende soorten tests.
Alle unit en integratie tests bevinden zich op een testing branch van de GitHub van het project. Deze branch wordt up to date gehouden met de meest recente

code.

5.1.2 Black-box tests

Onder de black-box tests vallen alle test procedures die geen kennis hebben van de interne werking van het systeem. Hun hoofddoel is het testen van de functionaliteiten. Hieronder vallen:

Component interface tests testen de handeling van data tussen verschillende modules van de applicatie

Systeem tests verifiëren dat de applicatie voldoet aan de vereisten

Acceptatie tests worden uitgevoerd door de cliënt.

Deze tests maken weinig of geen gebruik van test platformen, maar worden getest door actief gebruik van (een deel van) de applicatie.

5.2 Test procedures

Unit tests vormen de basis van het gehele test proces en zullen dus blijvend worden uitgevoerd gedurende de ontwikkeling van het systeem. Dit betekent dat elke methode zijn unit test zal verkrijgen, en deze blijft bestaan tijdens het evolueren van de methode. Telkens wanneer een methode wordt gewijzigd zal de bijhorende test opnieuw worden uitgevoerd. Zo wordt de correcte werking van iedere methode verzekerd en kan men gemakkelijker achterhalen wat er fout loopt bij het falen van een integratie test. De eerste unit tests worden geschreven voordat het ontwerp, beschreven in het SDD, wordt geïmplementeerd. Elke klasse beschreven in het SDD zal voor de beduidende methodes een test functie bevatten die de correcte werking van de methode zal verzekeren.

Het schrijven van de tests gebeurd met de Should syntax en wordt geplaatst in de test directory. De tester kan de test manueel oproepen of kan gebruik maken van het Mocha framework om alle testen met een commando uit te voeren. Een derde methode van testen is wanneer code naar GitHub (zie SCMP) wordt geduwd. Hierbij wordt de correcte functie opgeroepen en zal afhankelijk van het resultaat van de test de code op GitHub staan. Naast manuele uitvoering van de testen, zal ook bij elke commit op de test branch elke testfile worden uitgevoerd door het gebruikte build systeem (zie SCMP).

Code die door de testen is gecontroleerd, en door de QA Manager is gevalideerd mag op de master branch worden gepusht. Dit is echter een lang proces en maakt gebruik van Git's pull request systeem [5].

5.2.1 Problemen in testing

In het geval van een ontbrekende test, zal beslist worden wat er moet gebeuren afhankelijk van het type van de ontbrekende test. In het geval van een unit test zal de persoon die de methode heeft geschreven zelf verantwoordelijk zijn

[5] https://help.github.com/articles/using-pull-requests/

voor het schrijven van de test, ofwel wordt er beslist dat de Test Manager de taak op zich neemt om de test zo snel mogelijk te implementeren. In geval van een ontbrekende integratie test zal de Test Manager gecontacteerd worden en is deze verplicht zo snel mogelijk de test te schrijven, zodat de implementatie van bijhorende modules zo kort mogelijk moet worden stil gelegd.

Als een test faalt wordt de implementatie van de methode gecontroleerd op correctheid, dit aangezien dit de functie van de test is. Indien de methode volgens de implementator correct is, zal de Test Manager de test controleren en indien nodig aanpassen. Dit kan gebeuren wanneer de interface van een methode wordt gewijzigd.