Testbericht der Qualitätssicherungsphase

Definition und Durchführung von Messwertverarbeitung für den Physikunterricht auf Basis eines Raspberry Pis

Version 0.0.1

David Gawron Stefan Geretschläger Leon Huck
Jan Küblbeck Linus Ruhnke

25. August 2019

Inhaltsverzeichnis

1 Ziel des Testberichts

Das Ziel des Testberichtes ist es dem Leser einen Überblick über die verwendeten Testverfahren zu geben und die während der Qualitätssicherungsphase entdeckten Fehler zu dokumentieren. Die Qualitätssicherungsphase hat das Ziel, möglichst viele Fehler aufzudecken, diese zu korregieren und zu dokumentieren. Zusätzlich soll das unbemerkte Wiederauftreten bereits gefundener Fehler durch Regressionstests verhindert werden. Dabei werden die Funktionalitäten und deren Qualitäten getestet.

1.1 Bedingungsüberdeckung

Wir streben eine mehrfache Bedingungsüberdeckung an. Dadurch werden Zweig- ,Anweisungs-, einfache und minimal-mehrfache Bedingungsüberdeckung subsumiert. Eine einfache Bedingungsüberdeckung ist subsumiert nicht einmal die Anweisungsüberdeckung und ist somit ungeeignet. Eine minimal-mehrfache Bedingungsüberdeckung wäre ein guter Kompromiss zwischen Aufwand und Nutzen, allerdings verwendet unser Plug-In *EclEmma* für *JaCoCo* standardmäßig mehrfache Bedingungsüberdeckung. Außerdem ist die Anzahl an Bedingungen in unserer Anwendung noch überschaubar. Eine Pfadüberdeckung streben wir nicht an, da dessen Aufwand mit 2 hoch k skaliert, wobei k die Anzahl an Anweisungen ist.

2 Planung der Qualitätssicherungsphase

Die Qualitätssicherungsphase wird in drei Meilensteine aufgeteilt, siehe dazu Abbildung 1. Der erste Meilenstein wird erfüllt, wenn das Modul Model der Anwendung eine hohe Testüberdeckung erreicht. Dabei sollen alle Tests automatisch mit J-Unit ablaufen. Das Model ist die Basis, die alle anderen Module benutzen und auch diese verbindet. Deshalb ist die erste Priorität eine getestetes Modul, um komplexe Folgefehler für die anderen Module zu verhindern.

Im zweiten Meilenstein werden alle anderen Module, außer der GUI, getestet. Auch hier erfolgt das Testen über automatische J-Unit Tests.

Die GUI ist ein Sonderfall beim Testen, da diese nur sehr begrenzt mit automatischen Tests getestet werden kann. Deshalb wird diese im dritten Meilenstein getestet. Der Dritte Meilenstein umfasst die GUI und auch das Testen der gesamten Anwendung. Die GUI wird hauptsächlich über Klickstrecken getestet. Die gesamte Anwendung wird durch Testszenarien aus dem Pflichtenheft geprüft. Weiter werden Qualitätsanforderungen der Anwendung durch verschiedene Tests geprüft. Schließlich wird die Leistung und auch die Hardware für die Anwendung getestet.

Testplan für die Qualitätssicherung	11. bis 16. August	16.08.19	20.08.19	21.08.19	22.08.19	bis	25.08.19	26.08.19	27.08.19	28.08.19	29.08.19	30.08.19	31.08.19
	MS 0: Altlasten	**Meilenstein 1**: Model 90 % Abdeckung mit J-Unit			**Meilenstein 2**: Backend, Cache, Controller und Fileservice 90% Abdeckung mit J-Unit			Puffer	**Meilenstein 3**: GUI- Abdeckung, Belastungs- und Integrationstests				Puffer
Spalteninformationen		**Modul-Abdeckung Sollwert in %**			**Modul-Abdeckung Sollwert in %**				**Modul-Abdeckung Sollwert in %**				
Controller				0	0-30	30-60	95					95	
FileService & Main				0	0-30	30-60	95					95	
GUI				0					0-20	20-40	40-65	85	
Klickstrecken													
Monkey Testing & Hallway Testing													
Laufzeit & Speicherverbrauch													
Qualitätsanforderungen													
TestSzenarien													
Model			65	90			90					95	
Backend				0	0-30	30-60	90					95	
Cache				0	0-30	30-60	90					95	
Hardwaretest Sensoren													

Abbildung 1: Der Sollpan für die Qualtätssicherungsphase.

3 Gefundene Fehler und deren Regressionstests

Dieses Kapitel umfasst die Regressionstests für gefundene und behobene Fehler. Die Tests sind nach Modul und Klassen strukturiert. Jeder Regressionstest verweist auf ein Issue der verwendeten Bugtracking-Software (hier GitHub).

3.1 Model

3.1.1 Measurement Configuration

Issue Nr.7 in der Methode getInChan

Fehlersymptom: Unbehandelte NullPointer Exception bei Eingabe einer ungültigen Id.

Fehlerursache: Prüfen nach NullPointer Exception fehlt.

Fehlerbehebung: Eine Null Prüfung wurde implementiert.

Verantwortlicher: David Gawron

Issue Nr.8 in der Methode getOutChan

Fehlersymptom: Unbehandelte NullPointer Exception bei Eingabe einer ungültigen Id.

Fehlerursache: Prüfen nach NullPointer Exception fehlt.

Fehlerbehebung: Eine Null Prüfung wurde implementiert.

Verantwortlicher: David Gawron

Issue Nr.9 in der Methode addConnection

Fehlersymptom: Unbehandelte NullPointer Exception bei Eingabe einer ungültigen Id.

Fehlerursache: Prüfen nach NullPointer Exception fehlt.

Fehlerbehebung: Eine Null Prüfung wurde implementiert.

Verantwortlicher: David Gawron

Issue Nr.10 in der Methode removeConnection

Fehlersymptom: Unbehandelte NullPointer Exception bei Eingabe einer ungültigen Id.

Fehlerursache: Prüfen nach NullPointer Exception fehlt.

Fehlerbehebung: Eine Null Prüfung wurde implementiert.

Verantwortlicher: David Gawron

Issue Nr.11 in der Methode createInChannelList

Fehlersymptom: Auftreten einer Index Out Of Bounds Exception.

Fehlerursache: Eine Prüfung, ob der Index groß genug ist, fehlt.

Fehlerbehebung: Der Fehler wird abgefangen durch einen Vergleich der Anzahl der InChannel zwischen yaml-File und Prototypblock.

Verantwortlicher: David Gawron

Issue Nr.12 in der Methode getOutChanPosi

Fehlersymptom: NullPointer Exception beim Laden einer Messkonfiguration mit ungültigen Block Id.

Fehlerursache: Prüfen nach NullPointer Exception fehlt.

Fehlerbehebung: Es wird nach Null geprüft. Dann ergab sich eine Folgefehler, der sich in der Methode createLoadedConnections als eine Index Out Of Bounds Exception äußerte. Durch das Implementieren einer Methode checkBlockInitId, die prüft, ob eine geladene Id auch gültig ist, wurde der Folgefehler behoben.

Verantwortlicher: David Gawron

Issue Nr.13 in der Methode createInChannelList

Fehlersymptom: NullPointer Exception bei ungültiger Messkonfiguration mit einer fehlenden BlockChannelliste.

Fehlerursache: Prüfen nach NullPointer Exception fehlt.

Fehlerbehebung: Eine Prüfung nach Null wurde hinzugefügt.

Verantwortlicher: David Gawron

Issue Nr.18 in der Methode removeBlock

Fehlersymptom: Der Versuch einen nicht existierenden Block zu entfernen, resultiert in eines Index Out Of Bounds Exception.

Fehlerursache: Der Index wurde nicht geprüft.

Fehlerbehebung: Eine Prüfung des Indexes wurde hinzugefügt. Außerdem wurde der Rückgabewert der Methode von void zu boolean geändert.

Verantwortlicher: David Gawron

Issue Nr.19 in der Methode removeBlock

Fehlersymptom: Der Versuch eine Konfiguration ohne eine Liste von Block Ids zu laden, führt zu einer Null Pointer Exception.

Fehlerursache: Es wurde nicht nach Null geprüft.

Fehlerbehebung: Die betreffende Zeile wurde in einen schon existierenden Null-Check verschoben.

Verantwortlicher: David Gawron

Fehler Nr.35 in der Methode getInitId

Fehlersymptom: Die Methode funktionierte nicht richtig und gab immer NULL zurück.

Fehlerursache: Der Zugriff auf die Blöcke in der Hasmap der Konfigurationsblöcke schlägt fehl.

Fehlerbehebung: Die KonfigurationsId wird nun über die Blockliste der Messkonfiguration geholt.

Verantwortlicher: David Gawron

3.2 Cache

3.3 Backend

3.4 Controller

3.5 Fileservice und Main

3.6 GUI

4 Testen der GUI

4.1 Testen der GUI durch Klickstrecken

4.2 Monkey Testing

5 Testen der Qualität

5.1 Hallway Usability Testing

5.2 Testen der Qualität der Funktionalitäten

6 Durchführen der Testfälle aus dem Pflichtenheft

6.1 T010 Starten der Anwendung und Hilfe

Strukturelement	Beschreibung
Testfallnummer (Pflichtenheft)	T10
Testfallverweis	z.B. Pfad zur Testdatei und Name des Tests dort
(optional) Subunit-tests	
Verantwortlicher Tester	to do
Vorbedingung	to do
Nachbedingung	to do
Beschreibung und Testziel	ggfs. nicht durchführbar weil
Erwartetes Ergebnis	to do
Verhalten im Fehler-fall	to do
Getestete Anforde-rungen	to do

6.2 T020 Starten der Demo

6.3 T030 Lehrer erstellt und speichert eine Messkonfiguration

6.4 T040 Schüler bearbeitet Aufgabe

6.5 T050 Schüler startet Messung und speichert Ergebnisse

6.6 T200 Laden einer ungültigen Datei als Messkonfiguration

6.7 T210 Starten einer ungültigen Messkonfiguration

6.8 T220 Entfernen eines Sensors bei laufender Messung

7 Hardware Tests und sonstige Tests

7.1 Leistung und Speicherverbrauch

7.2 Hardware Test der Sensoren

7.3 Testen auf verschiedenen Systemen

8 Glossar

EclEmma EclEmma ist ein Plug-In für Eclipse für Code-Überdeckungsanalysen. Es basiert auf JaCoCo. Die hier verwendete Version ist 3.1.2.

JaCoCo JaCoCo ist eine freie Code-Überdeckungs Bibliothek für Java. Hier verwendete Version: 0.8.4.