



UNIVERSITÄT  
LEIPZIG

## Softwaretechnikpraktikum

---

# Testbericht

---

<b>Gruppe:</b>	nw19a
<b>Mitglieder:</b>	Thomas Pause, Sabine Lorus, Arik Korte, Martin George, Josephine Lange, Esther Prause, Anh Kiet Nguyen, Bärbel Hanle
<b>Verantwortlich:</b>	Thomas Pause
<b>Betreuer:</b>	Dr. Nicolas Wieseke
<b>Tutor:</b>	Martin Frühauf
<b>Abgabedatum:</b>	27.01.2020
<b>Version:</b>	0.6 (Release 2)

**Stand:** 27. Januar 2020

**Inhaltsverzeichnis**

<b>1</b>	<b>Verwendete Testframeworks</b>	<b>1</b>
<b>2</b>	<b>GitLab CI</b>	<b>1</b>
<b>3</b>	<b>Unit-Tests</b>	<b>1</b>
<b>4</b>	<b>Integrationstests</b>	<b>2</b>
<b>5</b>	<b>UI- und Systemtests</b>	<b>2</b>
<b>6</b>	<b>Manuelle Tests</b>	<b>3</b>
<b>7</b>	<b>Performance-Testing</b>	<b>3</b>
<b>A</b>	<b>Anhang</b>	<b>4</b>
	A.1 Testfälle der manuellen Tests . . . . .	4

## 1 Verwendete Testframeworks

Für die Durchführung der Komponententests verwenden wir JUnit5, für die Integrationstests die zugehörige Erweiterung Mockito.

Systemtests, insbesondere die korrekte Funktionsüberprüfung des User-Interfaces und Ende-zu-Ende-Tests werden mit Espresso realisiert. Hierfür wird teilweise der Espresso Test Recorder verwendet.

Des Weiteren haben wir die `gradle.build`-Dateien auf App- und Projektebene so angepasst, dass zum Beispiel die Ergebnisse der Tests übersichtlicher erkennbar sind, indem auch bei bestandenen Tests die jeweiligen Bezeichner ausgegeben werden.

Die sehr nützliche Funktion von Kotlin, Testnamen als Strings mit Leerzeichen definieren zu können, sorgt für bessere Lesbarkeit.

## 2 GitLab CI

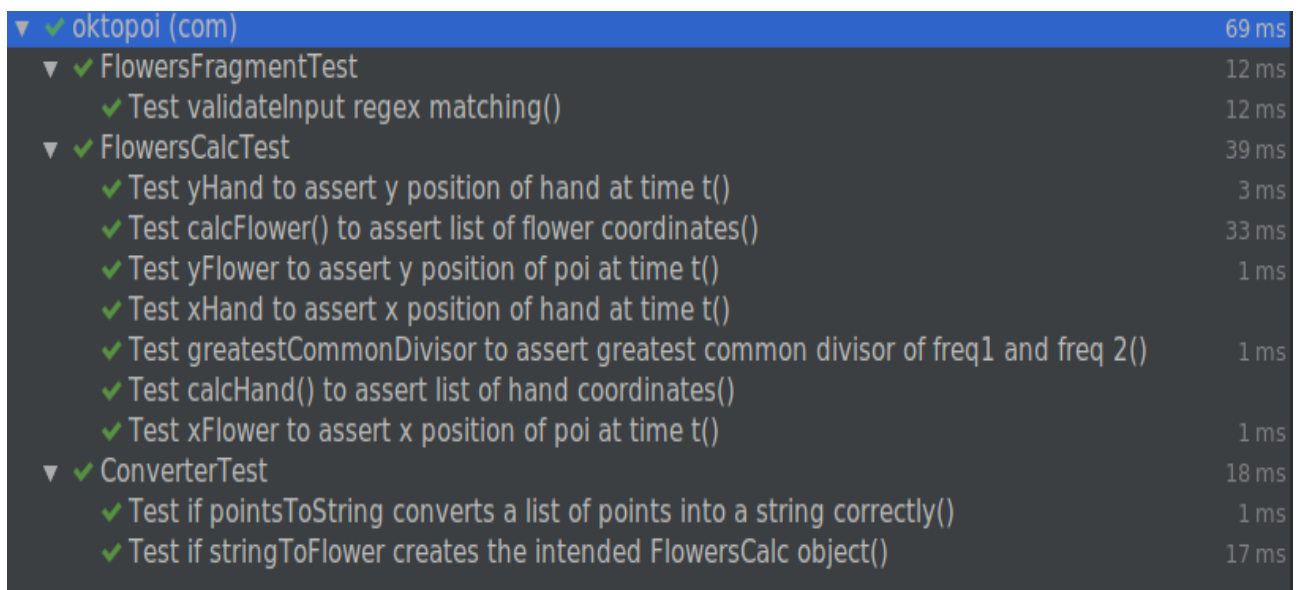
Wie im Qualitätssicherungskonzept vereinbart, werden bei jedem Push die drei Stages der GitLab Continuous Integration durchlaufen. Diese wurden in der Datei `.gitlab-ci.yml` definiert.

Es wird ein Build, das Linting und die Tests durchgeführt.

Dabei gibt es unterschiedliche Jobs für die verschiedenen Branches, welche mit Hilfe des *Gradle-Wrappers* durchgeführt werden.

## 3 Unit-Tests

Mit den Komponententests prüfen wir wichtige Standard- und Grenzfälle ab. Hierfür wurden in bislang 3 Testklassen verschiedene Tests entworfen. Die Parameter der zahlreichen Testfälle sind in den jeweiligen Testklassen ersichtlich. Im folgenden findet sich eine Auflistung der Testmethoden und ihr jeweiliges Ergebnis.



▼ ✓ oktopoi (com)	69 ms
▼ ✓ FlowersFragmentTest	12 ms
✓ Test validateInput regex matching()	12 ms
▼ ✓ FlowersCalcTest	39 ms
✓ Test yHand to assert y position of hand at time t()	3 ms
✓ Test calcFlower() to assert list of flower coordinates()	33 ms
✓ Test yFlower to assert y position of poi at time t()	1 ms
✓ Test xHand to assert x position of hand at time t()	
✓ Test greatestCommonDivisor to assert greatest common divisor of freq1 and freq 2()	1 ms
✓ Test calcHand() to assert list of hand coordinates()	
✓ Test xFlower to assert x position of poi at time t()	1 ms
▼ ✓ ConverterTest	18 ms
✓ Test if pointsToString converts a list of points into a string correctly()	1 ms
✓ Test if stringToFlower creates the intended FlowersCalc object()	17 ms

Da die aktuellen Unit-Tests auf nativem JUnit5 basieren, gibt es in diesem Release keine html-Berichte, weil dieses Feature nicht unterstützt wird. Stattdessen befindet sich ein Screenshot mit den Testergeb-

nissen im Releasepaket.

Über die weitere Nutzung von JUnit5 bzw. einen kompletten Wechsel zurück zu JUnit4 muss erst noch im Team abgestimmt werden.

## 4 Integrationstests

Zum aktuellen Entwicklungsstand sind noch keine Integrationstests nötig. Dennoch haben wir uns bereits mit dem Konzept des Mockings mithilfe der JUnit5-Erweiterung *Mockito* befasst und an einem Dummy-Projekt geübt.

## 5 UI- und Systemtests

Diese Tests wurden mit dem Framework *Espresso* durchgeführt. Sie benötigen eine virtuelle oder physische Android-Instanz (also einen aktiven Emulator oder ein Endgerät) und werden daher aus Performance-Gründen nicht über die GitLab-CI automatisch ausgelöst.

Nach der Umstrukturierung der Anwendung als *Single-Activity-App* und der Integration des *Navigation Drawers* wurden auch neue Systemtests und Tests der Benutzerschnittstelle nötig.

Momentan beinhaltet unsere App drei *Fragments*, für die Erstellung und Speicherung von Flowers sowie von Eingaben über das Touchpad und die Bibliothek, die die Dateien verwaltet und speichert. Daher bieten sich insgesamt 4 Tests an, jeweils einer pro individuellem Fragment und ein globaler Ende-zu-Ende-Test, der die möglichen Wege durch die Anwendung simuliert und das erwartete Verhalten validiert.

Die gewählten Testszenarien wurden am Anfang der jeweiligen Testklassen dokumentiert und können am angeschlossenen Endgerät oder am Emulator visuell verfolgt werden.

Dass Espresso JUnit4 benötigt, unterstreicht die Notwendigkeit einer Abstimmung, ob wieder komplett darauf umgestiegen werden sollte (siehe oben).

Im Folgenden sind die Ergebnisse der Systemtests zu sehen, als Ausschnitt aus der im Browser dargestellten Datei `index.html` im Ordner `reports/androidTests/connected/`.

### Package com.oktopoi

[all](#) > com.oktopoi

**4** tests  
**0** failures  
**29.731s** duration

**100%**  
successful

#### Classes

Class	Tests	Failures	Duration	Success rate
<a href="#">DrawByHandTest</a>	1	0	5.997s	100%
<a href="#">End2EndTest</a>	1	0	17.200s	100%
<a href="#">FlowersTest</a>	1	0	3.124s	100%
<a href="#">LibraryTest</a>	1	0	3.410s	100%

## 6 Manuelle Tests

Zusätzlich zu den automatisierten Systemtests führten wir manuelle Tests durch, um die korrekte Funktionalität der Canvas zu überprüfen.

Als Referenz diene hierbei die Seite <https://www.desmos.com/calculator/bjqsd2veim>, auf der wir für 6 Settings die Ergebnisse gegenübergestellt haben (siehe Anhang).

Folgende Konfigurationen wurden getestet und im Reviewgespräch am 08.01.2020 genauer erläutert und diskutiert:

- Testfall 1: Radius: 0.4 freq1: -3 freq2: 7 offset: 0.0
- Testfall 2: Radius: 0.5 freq1: 1 freq2: 1 offset: 0.5
- Testfall 3: Radius: 0.5 freq1: 3 freq2: 1 offset: 0.5
- Testfall 4: Radius: 1.25 freq1: 5 freq2: -1 offset: 0.5
- Testfall 5: Radius: 1.5 freq1: 20 freq2: -13 offset: 0.0
- Testfall 6: Radius: 0.9 freq1: -1 freq2: 1 offset: 0.1

Anhand der Ergebnisse lässt sich die Funktionalität der Canvaszeichnung von Flowers klar erkennen.

## 7 Performance-Testing

Um die oben erwähnten Testverfahren zu ergänzen haben wir die verschiedenen SDKs des Online-Dashboards *Firebase* integriert. Dabei handelt es sich um ein Google-Tool, welches uns Informationen zu folgenden Fragestellungen gibt:

→ **Performance:** Die Leistung auf unterschiedlichen Geräten wird übersichtlich dargestellt. Hierbei können zum einen vordefinierte und zum anderen eigene *Traces* genutzt werden, um Teilbereiche zu analysieren.

→ **Absturz-Tracking:** Die *Crashlytics-SDK* erfasst Abstürze der App, analysiert diese und kategorisiert sie nach Ausmaß des Fehlers, Verbreitung und weiteren Parametern.

→ **Systemtests auf verschiedenen Geräten:** auf bis zu 5 physischen und 10 virtuellen Geräten können verschiedene Testszenarien (auch selbst definierte Espresso-Tests) durchgeführt und ausgewertet werden. So erreichen wir eine breitere Testcoverage auf Systemtest-Ebene.

Damit bietet Firebase uns ein mächtiges Werkzeug mit einem übersichtlichen Dashboard für verschiedene globale Systemauswertungen und -analysen.

Aus Performancegründen wurde Firebase allerdings während der Entwicklungsarbeit auf dem Develop-Branch des GitLab-Repositories deaktiviert.

Für den Master-Branch wurde die Firebase-Konsole neu eingerichtet und kann nun über die Mail-Adresse [nw0ktopoi@gmail.com](mailto:nw0ktopoi@gmail.com) aufgerufen werden. Die nötigen Anmelde-Informationen werden Betreuer und Tutor persönlich übergeben.

Im *Testlab* befindet sich bereits wieder ein Systemtest, der die komplette Anwendung auf Stabilität testet.

## A Anhang

### A.1 Testfälle der manuellen Tests

