UNIVERSITÄT
LEIPZIG

Softwaretechnikpraktikum

Testbericht

Gruppe:	nw19a
Mitglieder:	Thomas Pause, Sabine Lorius, Arik Korte, Martin George, Josephine Lange, Esther Prause, Anh Kiet Nguyen, Bärbel Hanle
Verantwortlich:	Thomas Pause
Betreuer:	Dr. Nicolas Wieseke
Tutor:	Martin Frühauf
Abgabedatum:	02.03.2020
Version:	0.8 (Release 4)

Stand: 16. März 2020

Inhaltsverzeichnis

1 Verwendete Testframeworks

Für die Durchführung der Komponententests verwenden wir JUnit4.
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

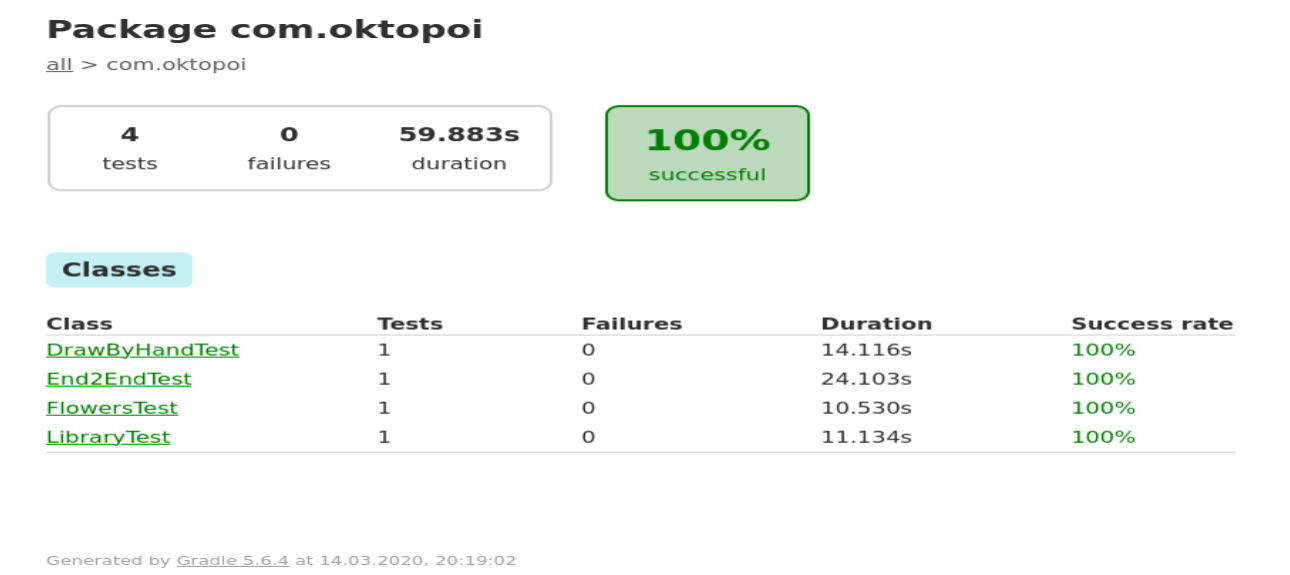
Mit den Komponententests prüfen wir wichtige Standard- und Grenzfälle ab. Hierfür wurden in 5 Testklassen verschiedene Tests entworfen. Die Parameter der zahlreichen Testfälle sind in den jeweiligen Testklassen ersichtlich. Im folgenden findet sich eine Auflistung der Testmethoden und ihr jeweiliges Ergebnis.

```
> Task :app:testReleaseUnitTest

com.oktopoi.ComplexTest

  Test test calcRadius PASSED
  Test test calcAngle PASSED
  Test test calcIm PASSED
  Test test calcReal PASSED

com.oktopoi.FlowersFragmentTest

  Test validateInput regex matching PASSED

com.oktopoi.FlowersCalcTest

  Test yHand to assert y position of hand at time t PASSED
  Test calcFlower() to assert list of flower coordinates PASSED
  Test xHand to assert x position of hand at time t PASSED
  Test yFlower to assert y position of poi at time t PASSED
  Test greatestCommonDivisor to assert greatest common divisor of freq1 and freq 2 PASSED
  Test calcHand() to assert list of hand coordinates PASSED
  Test xFlower to assert x position of poi at time t PASSED

com.oktopoi.ConverterTest

  Test if pointsToString converts a list of points into a string correctly PASSED
  Test if stringToFlower creates the intended FlowersCalc object PASSED

com.oktopoi.CurveToHandTest

  Test test calcHand() PASSED
  Test test fourier(freq) PASSED
  Test test mainCircle() PASSED

SUCCESS: Executed 17 tests in 767ms
```

Die ausstehende Entscheidung zwischen JUnit4 und JUnit 5 wurde im Team zugunsten von JUnit4 getroffen, sodass es in diesem Release auch wieder die generierten HTML-Berichte zum Nachschlagen gibt.

4 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 Drawer*s wurden auch neue Systemtests und Tests der Benutzerschnittstelle nötig.
Momentan beinhaltet unsere App fünf *Fragments*, für die Erstellung, Speicherung und Animation von Flowers sowie von Eingaben über das Touchpad sowie deren Animation. Desweiteren gibt es noch die Bibliothek, die die Dateien verwaltet und speichert und bereits vorgegebene Kurven enthält, sowie die Hilfe und die Poiwelt. Die letzten beiden sind eher statische Fragments, die lediglich Text enthalten und somit nicht einzeln getestet werden.
Daher bieten sich insgesamt 4 Tests an, jeweils einer pro funktionalem Fragment und ein globaler Ende-zu-Ende-Test, der alle 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.
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	59.883s duration	100% successful

Classes

Class	Tests	Failures	Duration	Success rate
DrawByHandTest	1	0	14.116s	100%
End2EndTest	1	0	24.103s	100%
FlowersTest	1	0	10.530s	100%
LibraryTest	1	0	11.134s	100%

Generated by Gradle 5.6.4 at 14.03.2020, 20:19:02

5 Manuelle Tests

5.1 Funktionalität der Canvas

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

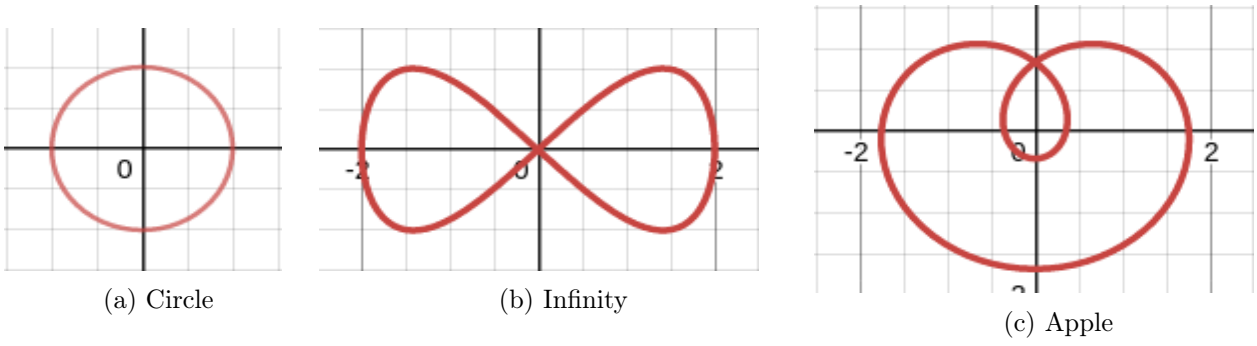
5.2 Berechnungen von Handkurven aus der Touchpadeingabe

In [1] wird beschrieben, wie wir mit Hilfe von komplexer Fouriertransformation aus einer gegebenen Kurve, die als Poibahn interpretiert wird, einen Vorschlag für die zugehörige Handbewegung generieren. Die dazu benötigten Funktionen wurden an drei Beispielkurven getestet (T gibt hier die Periodendauer an).

(a) **„circle“** $f(t) = \cos(\frac{\pi}{6}t) + i\sin(\frac{\pi}{6}t) = \exp(i\frac{\pi}{6}t) \qquad (T = 12)$

(b) **„infinity“** $f(t) = \sin\frac{\pi t}{12} + i\,\sin\frac{\pi t}{6} \qquad (T = 24)$

(c) **„apple“** $f(t) = \sin\frac{\pi t}{18} + \sin\frac{\pi t}{36} + i\,(\cos\frac{\pi t}{18} + \frac{2}{3}\cos\frac{\pi t}{36}) \qquad (T = 72)$



(a) Circle (b) Infinity (c) Apple

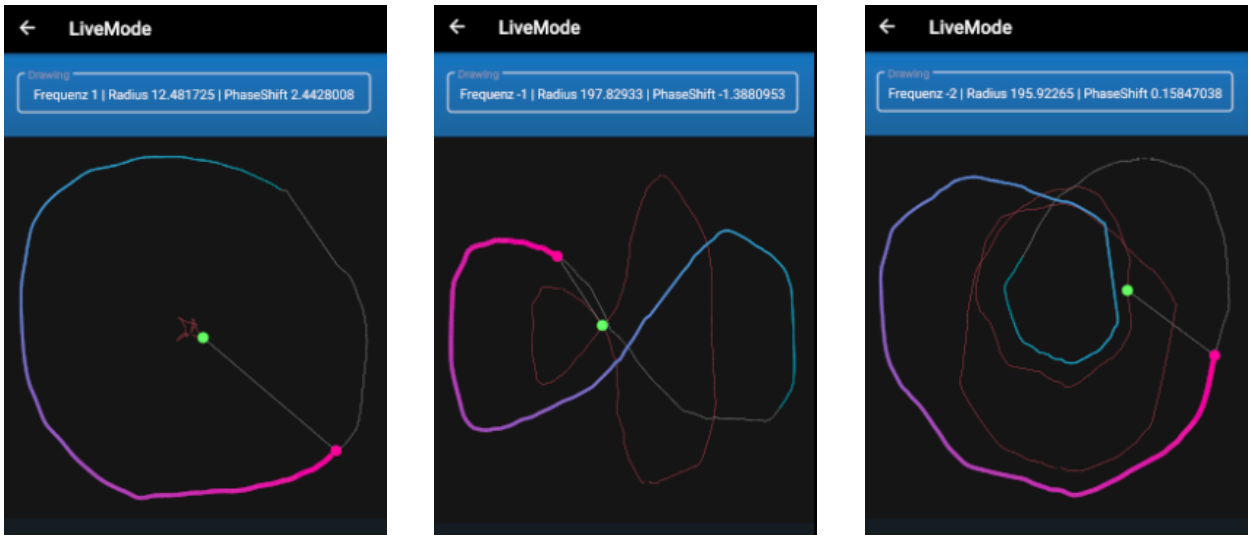
Abbildung 1: Testkurven für die CurveToHand-Klasse

Die für die Berechnung der Fourierkoeffizienten benötigten Integrale werden numerisch ausgerechnet. Wie in Tabelle 1 zu sehen ist, stimmen bei den Testkurven die analytisch berechneten mit den numerisch berechnenten Werten für die Fourierkoeffizienten überein.

Da das Programm als Eingabe bislang nur handgezeichnete Kurven zulässt, vergleichen wir hier die Testkurven mit ähnlichen Kurven, die durch Handeingabe erzeugt wurden. Obwohl diese recht zittrig ausfallen und große Abweichungen von den theoretischen Kurven aufweisen ergeben sich wie in Abbildung 2 zu erkennen ist durchweg die erwarteten Hauptfrequenzen.

Kurve	k	$\lvert c_k \rvert$ (numerisch)	$\lvert c_k \rvert$ (analytisch)	Hauptfrequenz
circle	-1	0	0	1
	0	0	0	
	1	1	1	
	2	0	0	
infinity	-3	0	0	-1
	-2	0.5	$\frac{1}{2}$	
	-1	1	1	
	0	0	0	
	1	1	1	
	2	0.5	$\frac{1}{2}$	
	3	0	0	
apple	-3	0	0	-2
	-2	1	1	
	-1	0.83333333	$\frac{5}{6}$	
	0	0	0	
	1	0.16666667	$\frac{1}{6}$	
	2	0	0	
	3	0	0	

Tabelle 1: Vergleich von numerisch und analytisch berechneten Fourierkoeffizienten



(a) circle (b) infinity (c) apple

Abbildung 2: Handgezeichnete Testkurven

6 Performance-Testing

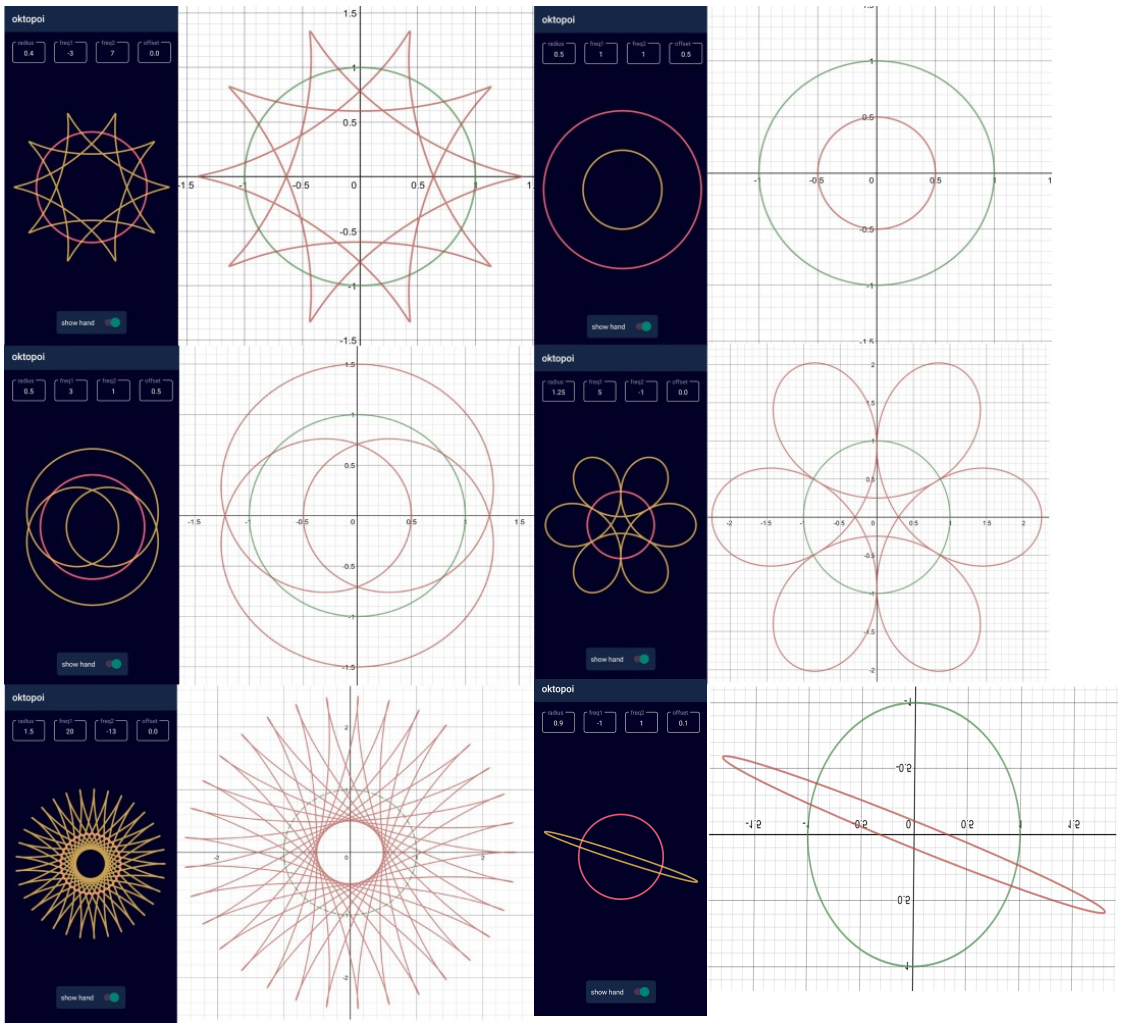
Um die oben erwähnten Testverfahren zu ergänzen haben wir die verschiedenen SDKs des Online-Dashboards *Firebase* integriert. Dabei handelt es sich um ein Google-Tool, welches uns Informationen zu folgenden Fragestellungen gibt:
→ **Performance:** Die Leistung auf unterschiedlichen Geräten wird übersichtlich 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 wird die Firebase-Konsole weiterhin genutzt, um Systemtests durchzuführen und Fehlfunktionen zu diagnostizieren.
Im *Testlab* befinden sich einige Robotests auf verschiedenen Geräten und in mehreren Sprachen, die die komplette Anwendung auf Stabilität testen.

A Anhang

A.1 Testfälle der manuellen Tests



Quellenverzeichnis

[1] *Berechnung der Handbewegung zu einer gegebenen Poibahn.* URL: `https://pcai042.informatik.uni-leipzig.de/~nw19a/Website/` (besucht am 28. 02. 2020).