UNIKASSEL
VERSITÄT

SOFTWARE ENGINEERING
UNIVERSITÄT KASSEL

GUI Testing und TestFX

Jens Kosiol

Testen grafischer Oberflächen

Das Testen grafischer Oberflächen hat viele Aspekte:

- Usability Aspekte: Wie gut ist die Anwendung benutzbar?
  - Sind Informationen gut auffindbar?
  - Lassen sich die Aufgaben, zu denen die Anwendung dient, (effizient) umsetzen?
  - …
  - Muss mit potentiellen Nutzern durchgeführt werden
- Funktionale Aspekte: Ist die intendierte Funktionalität korrekt umgesetzt?
  - Reagiert die Anwendung, wenn ein Button geklickt wird?
  - Werden Nutzereingaben ins Datenmodell übernommen?
  - …
  - Das Testen funktionaler Aspekte kann (teilweise) automatisiert werden.

In dieser Vorlesung konzentrieren wir uns auf das Testen funktionaler Aspekte.

Herausforderungen funktionaler GUI Tests

Das Testen grafischer Oberflächen stellt vor spezifische Herausforderungen:

- Nutzerinteraktion (Mausbewegungen, Klick, Texteingaben) muss emuliert werden.
- Testsituationen sind komplex und der Suchraum für Testfälle immens:
  - Was in einer spezifischen Situation passieren soll, hängt häufig von vorangegangenen Interaktionen ab ⇒ Test von Pfaden/Sequenzen statt von Einzelsituationen.
  - Es gibt viele mögliche Sequenzen von Nutzerinteraktionen.

Ansätze für GUI Tests

- Einsatz eines Testframeworks zur Automatisierung: ein Bot führt festgelegte Nutzerinteraktionen (Mausklicks, Eingabe von Text, …) mit der Anwendung durch.
- Auswahl von Testsequenzen:
  - Manueller Entwurf von Sequenzen von Interaktionen, die typische Benutzung der Anwendung abdecken
  - Randomisierung
  - Systematisches Generieren von Sequenzen durch Automaten, die Zustände der Anwendung beschreiben
  - Einsatz von Suchalgorithmen, um Pfade von Aktionen zwischen zwei Zuständen zu finden
  - …

In dieser Vorlesung entwerfen wir händisch Tests mit TestFX.

TestFX

- TestFX ist ein Framework zum Testen von JavaFX Anwendungen.
- Open Source Projekt: https://github.com/TestFX/TestFX
- Dokumentation der API: https://testfx.github.io/TestFX/docs/javadoc/
- Aktuelle Version: v4.0.17 (August 2023)
- Integrierbar mit JUnit
- Bots simulieren Nutzerinteraktion
- Unterstützung für headless testing (Ablauf der Tests ohne Einblenden der Anwendung)
- Leider kaum dokumentiert

Grundstruktur einer TestFX-Klasse

```
public class PartyAppTest extends
  ApplicationTest {

  @Override
  public void start (Stage stage)
      throws Exception {
    app = new App();
    app.start(stage);
  }

  @Test
  public void basicInteractionTest() {
    …
  }
}
```

Ausschnitt aus `PartyAppTest.java` (vereinfacht)

- Testklassen müssen von `ApplicationTest` erben.
- Die `start()`-Methode von `ApplicationTest` muss überschrieben werden; typischerweise wird hier die `start()`-Methode der eigentlichen Anwendung aufgerufen.
- Die einzelnen Testmethoden werden mit `@Test` annotiert.

FxRobotInterface und FxRobot

Die `FxRobotInterface`-Schnittstelle und die `FxRobot`-Klasse stellen Methoden zur Verfügung, um Nutzerinteraktion zu simulieren:

`clickOn(…)`	Methode, um (in Abhängigkeit von den übergebenen Parametern) Mausklicks durchzuführen
`doubleClickOn(…)`	Methode zum Durchführen von Doppelklicks
`press(…)`	Methode zum Anschlagen von Tasten
`rightClickOn(…)`	Methode zum Durchführen von Rechtsklicks
`write(…)`	Methode zur Texteingabe
…	

- Die Methoden stehen jeweils mit unterschiedlichen Parametern zur Verfügung.
- Die meisten Methoden geben das Objekt, auf das sie aufgerufen wurden, zurück (Unterstützung für „method chaining“).

Zugriff auf View-Elemente

```
@Test
public void basicInteractionTest() {
   clickOn("#partyNameTextField").write("SE BBQ");
   clickOn("#partyDateTextField")
      .write("21.06.24");
   clickOn("#createPartyButton");
}

@Test
public void selectButtonInitiallyDisabledTest() {
   Button selectButton =
      lookup("#selectExistingPartyButton").query();
}
```

Ausschnitt aus `PartyAppTest.java`

- Auch in TestFX kann man über gesetzte fx:id auf View-Elemente zugreifen.
- Viele der Roboter-Methoden liegen in einer Variante vor, in der eine solche ID als Parameter übergeben werden kann. Syntax: `"#<id>"`
- Wenn `Nodes` zurückgegeben (und nicht nur als Parameter benutzt werden sollen), steht die Klasse `NodeQuery` zur Verfügung:
  - Methoden wie `from(…)` oder `lookup(…)` geben eine `NodeQuery` zurück.
  - `query()` führt eine `NodeQuery` aus und gibt den ersten `Node` zurück, der dem spezifizierten Suchkriterium entspricht.

Assertions

In TestFX können JUnit Assertions verwendet werden, es wird allerdings die Verwendung der eigens zur Verfügung gestellten `verifyThat`-Methode empfohlen.

- Methode liegt in der Klasse `org.testfx.api.FxAssert`
- Typischerweise aufgerufen als `verifyThat(<Query>, <Matcher>)`
  - Der Matcher gibt alle Elemente zurück, die einem bestimmten Kriterium entsprechen.
  - `verifyThat` gibt genau dann true zurück, wenn der Knoten oder der Wert der Query unter den durch den Matcher extrahierten zu finden ist.
  - TestFX spezifiziert in den Paketen `org.testfx.matcher.base` und `org.testfx.matcher.control` Matcher für JavaFX-Elemente (Node, Button, Label, ListView, …)
  - Die Verwendung von Hamcrest (Framework zum deklarativen Definieren von Matchern) wird unterstützt.
  - Beispiele für Matcher: `is(…)`, `isEnabled()`, `hasText(…)`, `hasItems(…)`, …