DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

GUI TESTS

12. Oktober 2015

Lars Briem

(briem.lars@googlemail.com)

Duale Hochschule Baden Württemberg - Standort Karlsruhe

Gliederung

Gliederung

Motivation

Warum soll GUI getestet werden?

- Software muss getestet werden
- GUI ist Teil der Software
- GUI kann Fehler enthalten

Ziele

- Funktion von GUI Komponenten gewährleisten
- Zusammenspiel verschiedener Komponenten gewährleisten
- Test automatisieren (Continuous Integration)
  - Bei allen Änderungen
  - Auf allen Plattformen

GUI Tests

Funktionaler Test von GUI Komponenten

- Komponenten Tests
  - Testet Komponente einzeln
- Integrationstests
  - Testet Integration verschiedener Komponenten
- Akzeptanztests

$\Rightarrow$ Vergleichbar mit Unittests für Quellcode

$\Rightarrow$ Ähnliche Anforderungen wie an Unittests

GUI Tests

Funktionaler Test von GUI Komponenten

- Komponenten Tests
  - Testet Komponente einzeln
- Integrationstests
  - Testet Integration verschiedener Komponenten
- Akzeptanztests

$\Rightarrow$ Vergleichbar mit Unittests für Quellcode

$\Rightarrow$ Ähnliche Anforderungen wie an Unittests

Exkurs: Anforderungen an Unittests

- Automatisch
- Vollständig
- Wiederholbar
- Unabhängig
- Professionell
- Ergebnis: Bestanden oder Fehlgeschlagen

Gliederung

Arten

- Manuelles Testen
- Record und Replay
- Skriptbasiertes Testen
- Automatisiertes Testen

Manuelles Testen

Reale Person testet das Programm von Hand

- Protokoll mit Abfolge der Aktionen
- Ausführen einzelner Aktionen
- Überprüfen der Vorgaben
- Fehler oder Erfolg wird protokolliert

Manuelles Testen

+ Erkennung „sinnvoller“ Abweichungen der Vorgaben
+ Plausibilitätsprüfung der Vorgaben
+ „Gutes Aussehen“ bzw. Konsistenz überprüfbar
+ Schnell bei einmaliger Ausführung

− Extrem zeitaufwändig
− Extrem teuer
− Evtl. keine Wahrnehmung von Details
− Erfüllt Anforderung „automatisch“ nicht

Probleme skriptbasierter / automatischer Tests

- Suchen / Finden der GUI Elemente
- Interaktion mit den GUI Elementen
- Überprüfen der Ergebnisse
- Feststellen von Änderungen / Ereignissen
- Protokollierung von Fehlern

Suchen / Finden der GUI Elemente

- Position auf dem Bildschirm
- Beschriftung
- Elternelemente
- Attribute (z.B. CSS Klassen)
- Eindeutige Bezeichnung (z.B. fx:id)

Suchen / Finden der GUI Elemente

- Position auf dem Bildschirm
- Beschriftung
- Elternelemente
- Attribute (z.B. CSS Klassen)
- **Eindeutige Bezeichnung** (z.B. fx:id)

Interaktion mit den GUI Elementen - Robot

Zentrale Komponente jedes GUI Testtools

- Führt Benutzeraktionen aus
- Maussteuerung
  - Meist absolute Bildschirmkoordinaten
  - Move(x,y)
- Tastatureingaben
  - Einzelne Tastendrücke
  - Sequenzen von Tastendrücken
  - Tastenkombinationen
  - Press(„T“)
  - Press(Enter)
- Aufnahme von Screenshots

Überprüfen der Ergebnisse

- Vergleich der GUI mit Screenshot
  - Test bei identischem Screenshot bestanden
  - Aufwendig
  - Fehlerbehaftet
- Attribute der GUI Komponente auslesen
  - Möglichkeit zum Auslesen der Attribute notwendig
  - Einfache und präzise Möglichkeit
- Direkter Zugriff auf Businessmodell
  - Möglichkeit für Zugriff auf Businessmodell
  - Überprüfen der Effekte
  - Fehler in Businessmodell führt zu Fehler in GUI Test

Probleme skriptbasierter / automatischer Tests

Feststellen von Änderungen / Ereignissen

- Basierend auf Ereignissen
  - Beobachter an GUI Komponente registrieren
- Polling
  - Mehrfaches Abfragen eines Wertes
  - Abbruch bei Timeout

Protokollierung von Fehlern

- Wie bei Unit Tests
  - Bestanden
  - Fehlgeschlagen
- Screenshot

Record und Replay

- Älteste und einfachste Art der automatischen Tests
- Testerzeugung
  1. Aufzeichnung starten
  2. Programm starten
  3. Aktionen ausführen (Maus, Tastatur)
  4. Programm beenden
  5. Aufzeichnung beenden
  6. Tests / Überprüfungen definieren
- Erzeugter Test
  - Testskript
  - Grafischer Testablauf

Record und Replay - Beispiel Test

```
For i = 1 to 10000
    'VERIFY + AND * OPERATIONS ON THE CALCULATOR
    Window("Calculator").WinEdit("Edit").Set(i)
    Window("Calculator").WinButton("+").Click
    Window("Calculator").WinEdit("Edit").Set(i)
    Window("Calculator").WinButton("=").Click
    intResult_1 =   Window("Calculator").WinEdit("Edit").GetROProperty("text")

    Window("Calculator").WinEdit("Edit").Set(i)
    Window("Calculator").WinButton("*").Click
    Window("Calculator").WinEdit("Edit").Set("2")
    Window("Calculator").WinButton("=").Click
    intResult_2 =   Window("Calculator").WinEdit("Edit").GetROProperty("text")

    If intResult_1 <> intResult_2 Then
        Reporter.ReportEvent micFail, "VERIFY", "RESULT INCONSISTENT FOR DATA :" &i
    End If
Next
```

Record und Replay

Test enthält

- Schrittweise Anleitung
- Angabe der Aktionen mit
  - Absolute Pixelkoordinaten
  - GUI Komponenten (Objekte suchen)
- Manuell eingefügte Überprüfung
- Ausgabe im Fehlerfall

Record und Replay

- Testenablauf
  1. Testsoftware starten
  2. Test laden
  3. Ausführung der gespeicherten Schritte
  4. Aktionen und Fehler protokollieren
  5. Ergebnis abspeichern

- Fehlerfall
  - Viele Informationen sammeln (Screenshot, . . .)
  - Test abbrechen
  - Test weiter ausführen

Record und Replay

+ Einfach zu bedienen
+ Schnell für kleine Tests
+ Wenig Programmierkenntnisse notwendig

− Unübersichtlich bei komplexen Tests
− Nur manuelle Wiederverwendung
− Viel Redundanz
− Änderungen evtl. schwierig

Skriptbasiertes Testen

- Ähnlich wie Record und Replay
- Programmierer erstellt Skript

\+ Einfacher zu Warten

\+ Zusammenfassen verschiedener Aktionen

\+ Wiederverwenden von Standardaktionen

\+ Gleiche Programmiersprache wie GUI

− GUI bei Erstellung nicht unbedingt sichtbar

Automatisiertes Testen

- Künstliche Intelligenz
- Monkey Tests
- Matrix Tests

Künstliche Intelligenz

- Vergleichbar mit Robotik
  - Menge von Zuständen
  - Menge von Aktionen
  - Startzustand
  - Endzustand
  - Graphsuche von Start- zu Endzustand
- Erzeugung von Testfällen
  - Graphsuche (Tiefen-/Breitensuche, A*, . . .)
  - Evolutionäre Algorithmen
  - . . .

Künstliche Intelligenz

+ Generierung vieler Testfälle möglich
+ Viele Wege zum Ziel testbar

− Komplexe Definition von Zuständen / Aktionen
− Komplexe Suche / Optimierung

$\Rightarrow$ Bisher eher selten verwendet

Monkey Tests

- Simulation realer Anwender
- Zufällige Aktionsreihenfolge / -ausführung
  - Menü
  - Shortcut
  - Kontextmenü
- Aktionsfolge abspeichern
- Kontrollierter Zufallsgenerator
- Monkey versucht das Programm kaputt zu machen

$\Rightarrow$ Bei gefundenem Fehler: Aktionsfolge als dauerhaften Test hinzufügen

Matrix Tests

Einsatzbereich

- Oft großer Datenbereich für Eingabe
  - Zahlen beim Taschenrechner
  - Alle Eingaben testen nicht / nur schwer möglich
- Kombinationen verschiedener Eingaben

Ablauf

- Auswahl der Parameter
- Definition Parameterwerte
- Werte und Kombinationen in Tabelle zusammenfassen

Matrix Tests - Beispiel Taschenrechner

Addieren	**0**	**1**	**2**	**Große Zahl**
0	0	1	2	Große Zahl
1		2	3	Große Zahl + 1
2			4	Überlauf
Große Zahl				Überlauf

Integer: Große Zahl = $2^{31} - 2$

Long: Große Zahl = $2^{63} - 2$

$\Rightarrow$ Erweiterung auf negative Zahlen möglich

Matrix Tests - Beispiel Taschenrechner

+ Einfache Abdeckung vieler Kombinationen
+ Reduktion der Redundanz

− Exponentielle Laufzeit
  - k Parameter
  - n Werte pro Parameter
  - n^k Kombinationen

Vor- und Nachteile von GUI Tests

+ GUI wie Code während dem Buildvorgang testen
+ Einmal Aufwand für kontinuierliche Tests
+ 1 Test für unterschiedliche Plattformen
+ Monkey Test vergleichbar mit normalem Benutzer
+ Screenshots

− Ergebnis genau spezifizieren
− Einmalige Überprüfung durch Mensch schneller

Probleme verglichen mit Unittests

- Deutlich längere Laufzeit
- Asynchrone Abarbeitung
  - Immer Multithreading
  - Timeouts verwenden
- Grafischer Desktop sinnvoll / notwendig
- Eingabe während Ausführung blockiert

Gliederung

GUI Tests am Beispiel von TestFX

- GUI Test Framework für JavaFX
- Komponenten von TestFX
  - Suche von Elementen
  - Maussteuerung
  - Tastatursteuerung
  - Wartezeiten und Timeouts
  - Überprüfungen

Suche von Elementen

Einzelne Komponente finden mit `GuiTest#find`

- Label / Text auf der Komponente
  - `find(„Label“)`
- CSS Selektor oder `fx:id`
  - `find(„.parent-node #node“)`
- Lambda Expression
  - ```
    find( (ListView list) ->
    list.getItems().size() > 10)
    ```

Mehrere Komponenten finden mit `GuiTest#findAll`

- Akzeptiert Matcher z.B. `hasLabel(„text“)`
    ```

Maussteuerung

- Bewegung `move( )`
  - Absolute / Relative Koordinaten
  - `Node`
  - Selektoren
- Drag-and-Drop `drag( ).via( ).to( )`
  - Wie Bewegung
- Klicken `click( )`
  - Wie Bewegung
  - Ohne Relative Koordinaten
  - An aktueller Position
- Scrollen `scroll( )`
  - UP / DOWN
  - Anzahl Scritte

Tastatursteuerung

- Mehrere Buchstaben
  - `type(„Some text“)`
- Einzelne / Mehrere Tasten
  - `push(ENTER, ...)`
- Taste halten
  - `hold(SHIFT)`
- Taste loslassen
  - `release(SHIFT)`

Wartezeiten und Timeouts

- Feste Wartezeit
  - `sleep(10, SECONDS)`
  - Verlangsamt Test evtl. unnötig

  $\Rightarrow$ Selten verwenden

- Warten auf Ereignis
  - `waitUntil(Node, Matcher)`
  - `waitUntil(Selektor, Matcher)`
  - Zusätzlich Timeout möglich

  $\Rightarrow$ Timeout sollte immer angegeben werden

Überprüfungen - Matcher

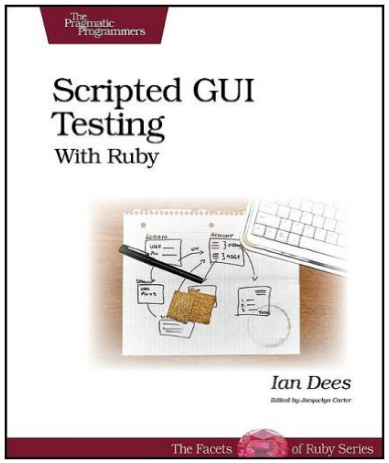
- Alle JUnit / Hamcrest Matcher
  - `is( )`
  - `equals( )`
  - `hasItems( )`
  - `contains( )`
  - `...`
- `hasLabel(„Some text“)`
- Properties überprüfen
  - `assertThat(myNode, hasText(„Some text“))`
  - `verifyThat(myNode, hasText(„Some text“))`
- Existenz von Elementen
  - `assertNodeExists(Matcher)`

Gliederung

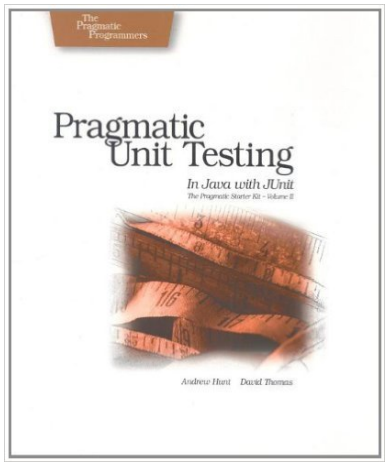
Live Demo

Gliederung

Literatur



- Scripted GUI Testing
  - Ian Dees
  - The Pragmatic Programmers
  - ISBN: 978-1934356180



- Pragmatic Unit Testing
  - Andrew Hund und David Thomas
  - The Pragmatic Programmers
  - ISBN: 978-0974514017

Weitere Quellen

- Internet
  - amazon.de
  - gettyimages.de
  - github.com/TestFX/TestFX
  - wikipedia.org