PM1/PT Ruby: Testen mit RUnit

Einteilung von Fehlern

Syntax Fehler

- Syntaxfehler sind Fehler, die durch falsche Verwendung von Sprachkonstrukten einer Programmiersprache entstehen. Z.B.
  1. nicht korrekt geschlossene Zeichen-ketten
  2. nicht korrekt geschlossene Blöcke (von Kontrollstrukturen, Klassen, Methoden, Iteratoren etc…)
  3. Methodenaufrufe mit falscher Pa-rameterzahl.
  4. falsche (nicht bekannte) Methoden oder Klassennamen

Erkennen von Syntaxfehlern

- Syntaxfehler sind leichter zu finden,
  - da sie teilweise bereits im Editor sichtbar werden. (gilt für 1. und 2.)
  - da die Fehlermeldungen, die zur Laufzeit auftreten, deutliche Hinweise auf die Fehlerursache geben. ***ArgumentError*** (zu 3.),

    NameError, NoMethodError (zu 4.)

Einteilung von Fehlern

Logische Fehler

- Logische Fehler liegen vor, wenn ein Programm ohne Fehlermeldung ausgeführt wird, aber Ergebnisse liefert, die nicht gewünscht sind.
- Logische Fehler sind sehr schwer zu entdecken. Bekanntermaßen finden wir in vielen kommerziellen Produkten logische Fehler.
- Es ist unmöglich die Korrektheit eines Programmes für den allgemeinen Fall zu beweisen.
- Daher ist es auch eine sehr anspruchsvolle Aufgabe, Tests für ein Programm zu entwickeln, die einen großen Teil logischer Fehler aufdecken.

Aufdecken logischer Fehler

- Werkzeuge, die uns dabei unterstützen, logische Fehler auf Modulebene aufzuspüren, sind die Unit-Test Umgebungen. (In Ruby **RUnit**).
- Diese Werkzeuge unterstützen uns darin , erwartete Ergebnisse von Programmen zu formulieren.
- Die schwere Arbeit, alle wichtigen zu testenden Fälle mit einem geeigneten Eingabesetting zu definieren, wird uns von diesen Werkzeugen **nicht** abgenommen!

Modultests

Modultest vs. Akzeptanztest

- Modultest engl. *unit test* bezieht sich auf das Testen einzelner Einheiten (Klassen, Methoden) einer Anwendung. Im Gegensatz dazu bezieht sich der Akzeptanztest auf die gesamte Anwendung.

- Modultest können fertiggestellt werden, bevor eine Methode oder Klasse geschrieben wird. Wenn wir ein sehr gute sprachliche Spezifikation vom Verhalten einer Klasse oder der Methoden haben, dann können auf dieser Basis bereits Modultests entwickeln.

- **Test Driven Development** ist eine moderne Entwicklungsmethode, in der das Schreiben von Testfälle vor die Entwicklung der Anwendung gesetzt wird.

Techniken und Werkzeuge

- Modultests mit **RUnit**
- Automatisierte Tests
- manuelle Ausführung
- Ausgaben auf der Konsole
- Debugger

Grenzfälle testen

- **Aufgabenstellung**: Erkennen von Palindromen

- Grenzfälle:
  - Leere Zeichenketten
  - Zeichenketten der Länge 1

- Das Programm muss auch für Grenzfälle korrektes Verhalten zeigen.

Ein guter Test enthält positive und negative Testfälle

Positives Testen:

- Wir testen die Fälle, die funktionieren sollten.

  Charakteristisch: Wir prüfen das Ergebnis der Methode ***ist_palindrom?*** auf **Erfolg** (***true***).

Negatives Testen:

- Wir testen die Fälle, die fehlerhaft sein sollten.

  Charakteristisch: Wir prüfen das Ergebnis der Methode ***ist_palindrom?*** auf **Misserfolg** (***not***).

- Auch im negativen Fall muss ein Programm in vorhersagbarer Weise fehlschlagen.

Regressionstests

- Es ist eine alte Erfahrung, dass sich durch Beheben von Fehlern in einem Programm neue Fehler einschleichen können.
- Um nach eine Änderung eines Programms sicher zu stellen, dass die ursprüngliche Funktionsweise erhalten bleibt, schreiben wir Testscripts, die nach jeder Änderungen auf das Programm angewendet werden.
- Diese Art des Testvorgehens ist unter dem Begriff **Regressionstest** bekannt.
- Mit den Ruby Scripten für die einzelnen Testszenarien können wir bereits mit einfachen Bordmitteln Regressionstests durchführen.

RUnit: Ein Werkzeug für das automatisierte Prüfen von Testergebnissen

- Ein Werkzeugs zur Unterstützung für den **Modultest.**

- unterscheidet zwischen
  - **Failures:** Fehler, die entstehen, wenn ein erwartetes Ergebnis nicht berechnet wird (logische Fehler)
  - **Error:** Fehlern, die auf falsches Bedienen von Methoden und andere Programmierfehler zurück-zuführen sind.

- Ein RUnit **Testscript wird nicht beendet**, wenn ein Fehler bei der Ausführung auftritt. Es ist sicher gestellt, dass alle Testmethoden unabhängig vom Fehler eines vorausgehenden Aufrufs einer Testmethode ausgeführt werden.

Die grundlegenden Konzepte von RUnit

- **RUnit** kennt zwei Fehlerarten:
  - **Failure**: eine Methode liefert nicht das erwartete Ergebnis
  - **Error**: eine Methode erzeugt einen Fehler (***Exception***), die auf einen Programmierfehler zurückzuführen ist.
- Die erwarteten Ergebnisse einer Methode werden als Behauptungen (**Assertions**) formuliert.
- Beispiele für Assertions:
  - ***assert (<meth_aufruf>) :*** prüft, ob ein Methodenaufruf ***true*** liefert.
  - ***assert_equal(<wert>, <meth_aufruf):*** prüft ob ein Methodenaufruf den erwarteten Wert ***<wert>*** zurückliefert.
- Die einzelnen Tests werden als Methoden eine Testklasse geschrieben. Diese muss von ***Test::Unit::TestCase*** ableiten.
- **RUnit** führt nur die Methoden der Testklasse aus, die mit dem Präfix ***test_*** beginnen.
- Es werden immer alle Testmethoden ausgeführt, auch wenn einzelne Methoden Fehler liefern.
- In der ***setup()*** Methode kann ein sogenanntes **Fixture** definiert werden. Das ist eine Menge von Instanzvariablen mit festen Werten. ***setup()*** wird vor jedem Aufruf einer Testmethode von **RUnit** ausgeführt, so dass die Instanzvariablen immer gleich initialisiert sind und alle Methoden mit den gleichen Testdaten starten.

Die erste eigene Testklasse

- **Übung**: Wir überlegen **RUnit** Tests für die Methode ***ist_palindrom?***.

- Vorgehen:
  1. Wir überlegen positive, negative Testfälle und Testfälle für die Grenzfälle
  2. Wir laden die Ruby Bibliotheken für den Test mit ***require "test/unit"***
  3. Wir erzeugen eine Rubyklasse und lassen diese von ***Test::Unit::TestCase*** ableiten.
  4. Wir überlegen das **Fixture**.
  5. Wir schreiben für jeden Test eine Testmethode, die mit **test_** beginnt.
  6. Wir formulieren die bei den Methodenaufrufen erwarteten Ergebnisse als **Assertions**.

- **Lösung**: Siehe Projekt ***V4-Runit*** Script ***zeichenketten_verarbeiten*** und das zugehörige TestScript

Zusammenfassung

- **Testen** ist das systematische Aufdecken von Fehlern in Programmen.
- **Syntaktische** Fehler entstehen durch falsches Benutzen von Sprachkonstrukten**. Logische Fehler** treten auf, wenn ein Programm falsche Ergebnisse liefert.
- **Modultests** sind Tests von Programmen, die einzelne Personen entwickelt haben. Sie prüfen, ob ein Programm bei definierten Eingaben erwartete Ergebnisse liefern.
- **Regressionstests** sind wiederholbare automatisierte Tests, die nach Änderungen die „alte“ Funktionalität überprüfen.
- **Positive** Tests prüfen die erwarteten Ergebnisse eines Methodenaufrufs. **Negative** Tests prüfen Fälle, die fehlerhaft sein sollten.
- **RUnit** ist ein Werkzeug zur Automatisierung der Testausführung. **Assertions** prüfen erwartete Ergebnisse von Methodenaufrufen. **Fixtures** definieren eine festen Testdatensatz für Testmethoden.
- **Failures** sind Fehler, die anzeigen, dass erwartete Testergebnisse nicht eintreten. **Errors** sind Programmierfehler.