Unittests für Einsteiger

Johannes Hubertz

SFD Köln, 19. September 2015

Motivation, Weshalb testen?

Ein wenig Theorie

Ein wenig Praxis

Fragen und Kritik

Warum wollen wir testen?

Wer redet von Softwarekrise?

- Komplexität steigt, Fehlerrate ist bestenfalls konstant
- Seit den 1980'ern ist die Rede von der Softwarekrise
- Fehlerfreiheit ist nicht herstellbar

Software-Engineering ist die Lösung (seit den 1990'ern?)

- Wasserfallmodell, V-Modell
- Anforderungsprofile, Pflichtenheft, Lastenheft
- Modularisierung, Ada, 4th-GL, ...
- Qualität wird besser, aber nicht gut

Und heute?

- Continous Integration, SCRUM, bedarfsorientierte Entwicklung ...
- Agile Methoden: Untested Software is broken by design ...

Richard Feynman: Method of problemsolving

- Write down the problem.
- Think hard!
- Write down the solution.

Was ist eine Unit?

Unit ist der kleinst mögliche zusammenhängende Teil des Codes,
der separat bzw. isoliert testbar ist.

Mathematik macht glücklich

Definition:

Sei $\mathbb{M}$ eine Menge.

Seien $\mathbb{N}_j \neq \emptyset \quad (j, k \in \mathbb{J})$ Untermengen von $\mathbb{M}$.

Dann sind die $\mathbb{N}_j$ eine **Partition** von $\mathbb{M}$, wenn

$$\mathbb{M} = \bigcup_{j \in J} \mathbb{N}_j \text{ } \textbf{und} \text{ } \mathbb{N}_j \cap \mathbb{N}_k = \emptyset$$

Noch Fragen?

Definition:

Eine **Äquivalenzrelation** (Bezeichnung: $\sim$) ist eine zweistellige Relation, für die folgende Regeln gelten:

1: $x \sim x$ (reflexiv)
   Jedes Element steht zu sich selbst in Relation.

2: $x \sim y \Rightarrow y \sim x$ (symetrisch)
   Wenn x zu y, dann steht auch y zu x in Relation.

3: $x \sim y \wedge y \sim z \Rightarrow x \sim z$ (transitiv)
   Wenn x zu y und y zu z, dann steht auch x zu z in Relation.

Satz: Sei $\mathbb{M}$ eine Menge

Sei $\sim$ eine Äquvalenzrelation auf $\mathbb{M}$. Für $m \in \mathbb{M}$ setzen wir:

$$[m] = \{m' | m' \in \mathbb{M}, m' \sim m\}$$

und nennen $[m]$ die Äquvalenzklasse von m (bezüglich $\sim$). Damit gilt:

$$\mathbb{M} = \bigcup_{m \in \mathbb{M}} [m]$$

Ferner ist

$$[m] \cap [m'] = \begin{cases} 0 & \text{für } m \nsim m' \\ [m] = [m'] & \text{für } m \sim m' \end{cases}$$

Sind $[m_j]$ mit $j \in \mathbb{J}$ die verschiedenen Äquivalenzklassen, so ist

$$\mathbb{M} = \bigcup_{j \in \mathbb{J}} [m_j]$$

eine Partition von $\mathbb{M}$.

Mathematik ist zu kompliziert?

Ein Beispiel:

Sei $\mathbb{M}$ eine Menge roter, gelber und grüner Kugeln.
Äquivalenzrelation: $x \sim y \iff$ Kugel x hat gleiche Farbe wie y.

1. `Reflexivität:` $f(x) = f(x)$
2. `Symmetrie:` $f(x) = f(y) \iff f(y) = f(x)$
3. `Transitivität:` $f(x) = f(y) \wedge f(y) = f(z) \implies f(x) = f(z)$

Äquivalenzklasse: $\mathbb{M} \supset \mathbb{G} = \{x | x \in \mathbb{M} \cap f(x) = \text{gelb}\} \Leftrightarrow$ alle gelben Kugeln.
Eine Partition besteht aus der Vereinigung **aller** so gebildeten Teilmengen aus je den roten, gelben und grünen Kugeln.

Alles klar?

Früher sprachen wir auch von Fallunterscheidung

Nur exakte Betrachtung ermöglicht exakte Lösung

Mathematik *kann* helfen, Formalisierung unterstützt

Die Aufgabenstellung: Implementieren Sie

Def.: Funktion signum(x), hier als sign(x) für $x \in \mathbb{R}$:

$$sign(x) = \begin{cases} 1 & \text{für } x > 0 \\ 0 & \text{für } x = 0 \\ -1 & \text{für } x < 0 \end{cases}$$

Wie kann das implementiert werden?

Wieviele Fehlermöglichkeiten sind dabei?

Nur mit den richtigen Tests wird **Zuverlässigkeit** erreicht

Was sind die richtigen Tests?

Anders gefragt, wie geht das?

Die Antwort auf alle Fragen: **Test Driven Development**

TDD: Wie geht das?

Vorgehensweise in mehreren Schritten

1. Umgebung definieren: def, Klasse oder Modul?
2. Äquivalenzklassen bestimmen
3. Grenzwerte feststellen
4. Prototyp schreiben ohne jeden Inhalt: **pass**

5a. mind. 1 Test schreiben pro Äquivalenzklasse

5b. Code zur Erfüllung der Tests in den Prototyp eintragen

5c. Tests laufen lassen bis zur Fehlerfreiheit

6. Ergänzende Tests mit fehlerhaften Eingabewerten schreiben

Merke:

Zuerst den Test, danach erst den Code schreiben.

Das schafft Klarheit im Denken...

Def.: Funktion signum(x), hier als sign(x) für $x \in \mathbb{R}$:

$$sign(x) = \begin{cases} 1 & \text{für } x > 0 \\ 0 & \text{für } x = 0 \\ -1 & \text{für } x < 0 \end{cases}$$

Schritt 1: Umgebung definieren

Der Einfachheit halber reicht hier ein def sign(x):

Schritt 2: Äquivalenzklassen bestimmen

Die Definition macht es uns einfach: $> 0, = 0, < 0$

Nicht vergessen: Ungültige Eingaben!

$\Longrightarrow$ Vier Äquivalenzklassen

console live show

Quellen und Hinweise

Python Testing Cookbook, Pact Publishing, 2011

Python Testing Beginners Guide, Pact Publishing, 2010

`https://github.com/rbreu/python-course`

`http://wiki.python-forum.de/pycologne/Protokoll20130410`

`http://www.python-course.eu/python3_tests.php`

`http://pythontesting.net/start-here/`

`https://github.com/gregmalcolm/python_koans`

Untested Software is broken by design

Ich bedanke mich für Ihre Aufmerksamkeit

hubertz-it-consulting GmbH jederzeit zu Ihren Diensten:

`verlässliche Netzwerke für vertrauliche Kommunikation`

Ihre Sicherheit ist uns wichtig!

Frohes Schaffen

Johannes Hubertz

`it-consulting _at_ hubertz dot de`

powered by python and LaTeX 2ε