Softwaretesten

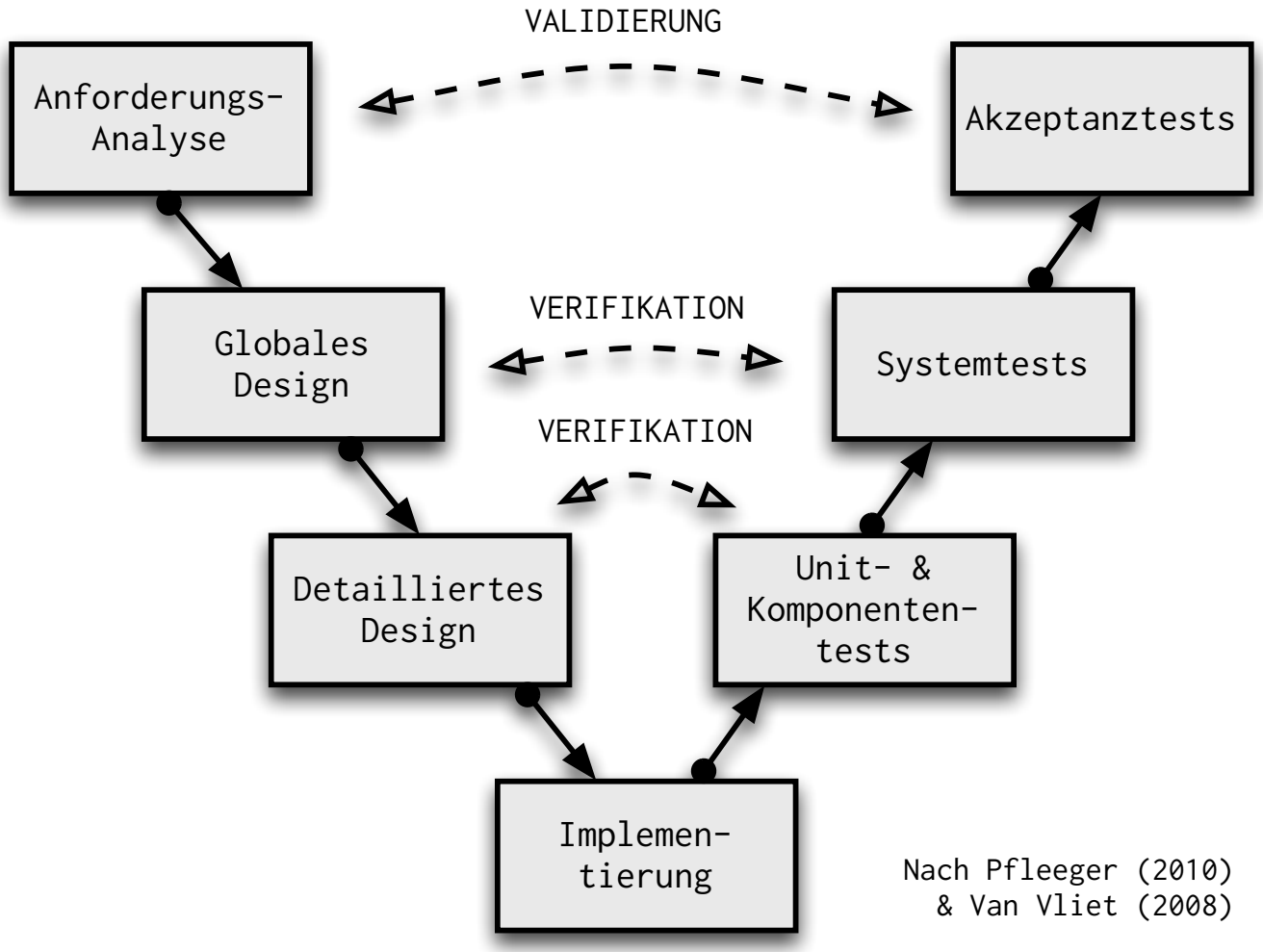
Hans Fangohr

June 2015

Was ist Softwaretesten?

- Bestandteil von Validierung und Verifikation
  - Validation: Are we building the *right* product? [Stellen wir das richtige Produkt her?]
  - Verification: Are we building the product *right*? [Stellen wir das Produkt richtig her? (Boehm, 1979)
- Vielseitige Tests von Code und Funktionalität, zB
  - ausführbarer Testcode (dynamische Tests)
  - Code und System Reviews (statische Tests)
- durch Entwickler, Testteams, Kunden

Testen im Softwarezyklus



Nach Pfleeger (2010)
& Van Vliet (2008)

Beispiel: V-Model Softwareentwicklungsprozess

Unittest Beispiel

Anforderung: *Funktion f, die ein Parameter n vom Typ integer entgegennimmt und die Summe der Zahlen von 0 bis n berechnet, und zurück gibt. Wenn n negativ ist, soll -1 zurück gegeben werden:*

$$f(n) = \begin{cases} \sum_{i=0}^{n} i, & \text{falls } n \geq 0 \\ -1, & \text{sonst} \end{cases}$$

$\rightarrow$ Beispiel

Beispiel: Implementierung

```
1  def f(n):
2      """
3      Funktion f nimmt n (Integer) und gibt zurueck:
4        fuer n >= 0: die Summe von 0 bis n
5        fuer n <  0: -1
6      """
7      if n >= 0:
8          summe = 0
9          for i in range(0, n+1):   # Schleife von 0 bis n
10             summe = summe + i
11         return summe
12     else:
13         return -1
```

Beispiel: Manuelles testen

```
$ python -i lib.py
>>> f
<function f at 0x1003bebf8>
>>> f(1)
1
>>> f(0)
0
>>> f(-10)
-1
>>> f(3)
6
>>> f(3) == 6
True
>>> f(3) == 0 + 1 + 2 + 3
True
>>> assert f(3) == 0 + 1 + 2 + 3
>>>
```

Automatisierung

- Wenn möglich sollten Tests so geschrieben werden, dass sie wiederholt ausgeführt werden können.

  $\rightarrow$ Beispiel

Beispiel: Testcode automatisiert

```
1  def test_f():
2      assert f(3) == 0 + 1 + 2 + 3
3      assert f(0) == 0
4      assert f(-10) == -1
```

Ausführen mit

```
py.test -v filename.py
```

Wie effektiv ist Testen?

"Testen kann nur Fehler aufweisen, nicht nachweisen, dass keine Fehler vorhanden sind." Dijkstra et al , (1972)

"Testing can only show the presence of errors, not their absence"

Teststrategien Unittest

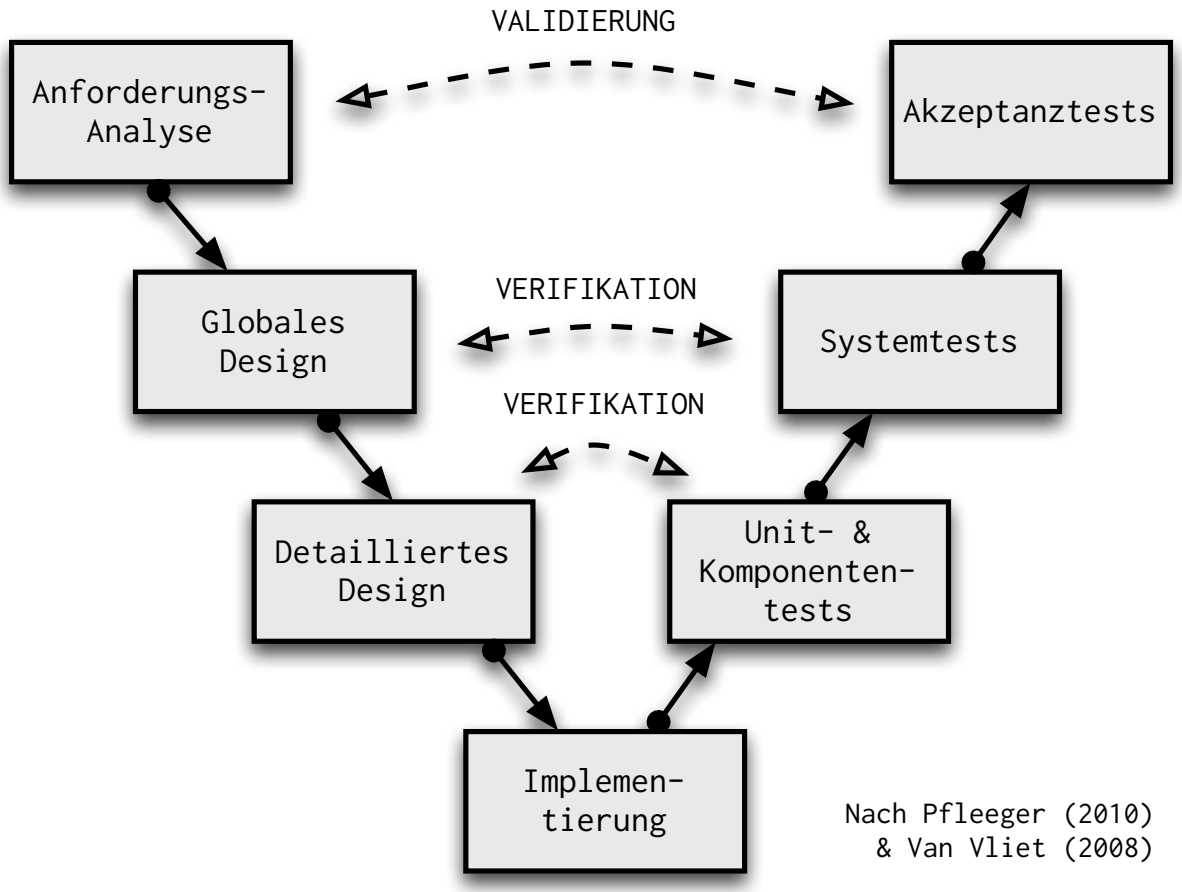
- Alle Fehlermeldungen auslösen
- Eingabedaten, die zu Underflow und Overflow führen
- Falsche Eingabedaten verwenden
- Zeilenabdeckung, Pfadabdeckung
- Partitionierung: Parameterklassen identifizieren, dann testen
  - innerhalb jeder Parameterklasse
  - an den Klassengrenzen

  $\rightarrow$ Beispiel
- ...

Beispiel: Testcode partitioniert Parameterraum

```
1  def test_f_positive():
2      assert f(3) == 0 + 1 + 2 + 3
3      assert f(10) == 55
4
5  def test_f_negative():
6      assert f(-20) == -1
7      assert f(-10) == -1
8
9  def test_f_grenzfall():
10     assert f(0) == 0
11     assert f(-1) == -1
```

Testen im Softwarezyklus



Beispiel: V-Model Software Engineering

- Traditionelles Bild: testen nach dem Implementieren
- Agile Methoden: testen während der Entwicklung
- eXtreme Programming (XP): zuerst den Test entwickeln, dann den Code
  → Test-Driven Development

Test-Driven-Development

- Test-Driven Development (TDD):
  1. Wähle Feature zum Implementieren
  2. Schreibe Testcode
  3. Schreibe eigentlichen Code, bis alle Tests bestehen
  4. Zurück zu 1
- Vorteile
  - Automatische Testabdeckung
  - Refaktoring leichter ($\rightarrow$ Beispiel)
  - Vereinfachtes Debuggen (Test zeigen Fehler auf)
  - Tests tragen zur Dokumentation bei

Ausblick: Testen in der Praxis

- Programmierer und Tester
- White box testing & black box testing
- Priorisieren: Repräsentatives Testen, Schwachstellen-orientiertes Testen, Risiko-orientiertes Testen
- Bug seeding
- Use-case basiertes Testen

Zusammenfassung

- Testen ist Teil des Qualitätsmanagements von Software
- Verifikation und Validierung
- Tests können nicht Abwesenheit von Fehlern beweisen
- Unittests, Komponententests, Systemstests, Abnahmetests, Regressionstests
- Dynamische und Statische Tests
- Beispiel: automatisierter Unittest (dynamisch, Verfikation)

Ausgewählte Referenzen

Bücher

- Van Vliet, Software Engineering, Wiley, 2008
- Somerville, Software Engineering, Pearson, 2010
- Pfleeger & Atlee, Software Engineering: Theory and Practice, Prentice Hall, 2009

Tools

- Testing: py.test, Java: JUnit (XUnit)
- Versionskontrolle: Git, Mercurial [Github, Bitbucket]
- Kontinuierliche Integration: Hudson/Jenkins [Travis CI]

Vorlesungsunterlagen und Beispielcode

- http://github.com/UP-softwareengineering/softwaretesten