Assertions (Zusicherungen)

Jens Dobberthin

April 10, 2005

Oberseminar Software-Entwicklung

Inhalt

1. **Einführung** (Motivation, Tony Hoare, Programmverifikation)
2. **Design by Contract** (Idee, Eiffel)
3. **Praxis: Programming by Contract for Python**
4. **Zusammenfassung und Diskussion**

Motivation

- Durch Testen eines Programms kann nicht sichergestellt werden, dass das Programm fehlerfrei ist!
- Unit-Tests beweisen nur, dass das Programm für vorher definierte Test-daten korrekt arbeitet.
- Alle bei einem Programm möglichen Eingabedaten zu testen, ist in der Regel unmöglich!
- *Kann man Methoden finden, die die Richtigkeit eines Programmes gewährleisten, ohne sich auf Tests verlassen zu müssen?*

Korrektheit von Programmen

Die Korrektheit eines Programmes lässt sich unter drei Aspekten betrachten:

1. Was bedeutet ein vorgegebenes Programm P? Dass heißt, wie lautet seine Spezifikationn S?

2. Entwickle Programm P, das die vorgegebene Spezifikation S implementiert.

3. *Führen Spezifikation S und Programm P die gleiche Funktion aus?*

Tony Hoare und Programmverfikation

- Charles Antony Richard Hoare; britischer Informatiker
  - Quicksort-Algorithmus
  - Hoare-Logik zum Nachweis der Korrektheit von Algorithmen
    - *An Axiomatic Basis for Computer Programming* (1969)
    - *Proof of Correctness of Data Representations* (1972)

- Prädikatenkalkül
  - Notation: *{P}S{Q}*
    - Wenn P vor der Ausführung von S wahr ist, dann ist Q nach der Beendigung von S wahr.
  - Inferenzregeln

Programmverifikation (Inferenzregeln)

Regel (Axiom)	**Prämisse**	**Folgerung**
1. Konsequenz 1	{P}S{Q},(Q $\Longrightarrow$ R)	{P}S{R}
2. Konsequenz 2	(R $\Longrightarrow$ P), {P}S{Q}	{R}S{Q}
3. Komposition	{R}S_1{Q},{Q}S_2{R}	{P}S_1;S_2{R}
4. Zuweisung	x := Ausd	{P(Ausd) x := Ausd {P(x)}
5. if	{P $\wedge$ B}S_1{Q}, {P $\wedge \neg$ B}S_2{Q}	{P} **if** B **then** S_1 **else** S_2 {Q}
6. while	{P $\wedge$ B}S{P}	{P} **while** B **do** S {P $\wedge \neg$ B}

- kann gezeigt werden, dass die Prämisse jeder Inferenzregel wahr ist, kann die Prämisse durch die Folgerung ersetzt werden
- axiomatische Verifikation erfolgt rückwärts, d.h. aus der bekannten Nachbedingung wird die Vorbedingung abgeleitet

Design by Contract

- von Bertrand Meyer [1] für die Programmiersprache Eiffel entwickelt
- Dokumentation und Verständigung von Rechten und Pflichten von Softwaremodulen, um die Korrektheit eines Programmes sicherzustellen
- Korrektheit?
  - Es tut das, was es behauptet zu tun!

Design by Contract (II)

Jede Funktion oder Methode in einem Softwaresystem macht etwas.

- **Vorbedingungen** beschreiben die Anforderungen, die vor Ausführung erfüllt werden müssen
- **Nachbedingungen** garantieren einen bestimmten Zustand nach der Ausführung
- **Klasseninvarianten** sind Bedingungen, die aus Sicht des Aufrufers immer gelten.

Design by Contract (Beispiel)

```
1
2  -- Add an item to a doubly linked list,
3  -- and return the newly created node.
4  add_item (item : STRING) : NODE is
5    require
6       item /= Void                        -- is not equal
7    deferred                               -- Abstract base class
8    ensure
9       result.next.previous = result       -- Check the newly
10      result.previous.next = result       -- added node's links.
11      find_item(item) = result            -- Should find it.
12   end
```

Design by Contract (III)

- *Wenn alle Vorbedingungen der Methode vom Aufrufer erfüllt werden, wird die Methode die Erfüllung aller Nachbedingungen und Invarianten nach ihrer Ausführung garantieren.*

- Bei Nichterfüllung der Vertragsbedingungen durch eine der Parteien entstehen Konsequenzen!
  - Ausnahmen

- Jede Verletzung des Vertrages ist ein Fehler!
  - Vorbedingungen dienen nicht der Überprüfung von Benutzereingaben!

Design By Contract (IV)

- als integraler Bestandteil der Programmiersprache
  - Eiffel
- durch Präprozessoren, die das Einbetten von Verträgen als spezielle Kommentare im Quelltext ermöglichen
  - iContract für Java [3]
  - Nana für C/C++ [2]
  - PyContract für Python [4]

Beispiel: Ringpuffer

- Elemente dürfen nur am Ende eingefügt und am Anfang entwert werden
- begrenzte Aufnahmemöglichkeit von Elementen
- Methoden
  - *init(leng)* - Initialisierung des Ringpuffers mit *leng* Elementen
  - *put(item)* - Hinzufügen eines Elementes
  - *get(item)* - Entfernen eines Elementes
  - *is_full()* - Überprüfung ob Puffer voll ist
  - *is_empty()* - Überprüfung ob Puffer leer ist

Beispiel: Konstruktur

```
1
2  def __init__(self, leng):
3      """Construct an empty circular buffer.
4
5          pre::
6              leng > 0
7          post[self]::
8              self.is_empty() and len(self.buf) == leng
9      """
10     self.buf = [None for x in range(leng)]
11     self.len = self.g = self.p = 0
```

Beispiel: *get(item)*

```
1
2  def get(self):
3      """Retrieves an item from a non–empty circular buffer.
4
5        pre::
6            not self.is_empty()
7        post[self.len, self.g]::
8            self.len == __old__.self.len – 1
9            __return__ == self.buf[__old__.self.g]
10     """
11     result = self.buf[self.g]
12     self.g = (self.g + 1) % len(self.buf)
13     self.len –= 1
14     return result
```

Beispiel: Test

- Test mit ohne Verletzung des Vertrages
- Test mit Verletzung des Vertrages
- Test mit einem weniger restriktiven Vertrag

Zusammenfassung und Diskussion

Design by Contract

- ✔ fördert ein besseres Verständnis der Funktionen, Methoden und Klassen
- ✔ ist ein systematischer Ansatz für die Entwicklung von fehler-freien objekt-orientierten Systemen
- ✔ ist eine Methode zur Dokumentation von Software
- ✔ ist eine Technik im Umgang mit unerwarteten Fehlerquellen (sicherer Umgang mit Ausnahmen)

References

[1] Bertrand Meyer: *Object-Oriented Software Construction.* Prentice Hall, 1988.

[2] http://www.gnu.org/software/nana/nana.html

[3] http://www.javaworld.com/javaworld/jw-02-2001/jw-0216-cooltools.html

[4] http://www.wayforward.net/pycontract/