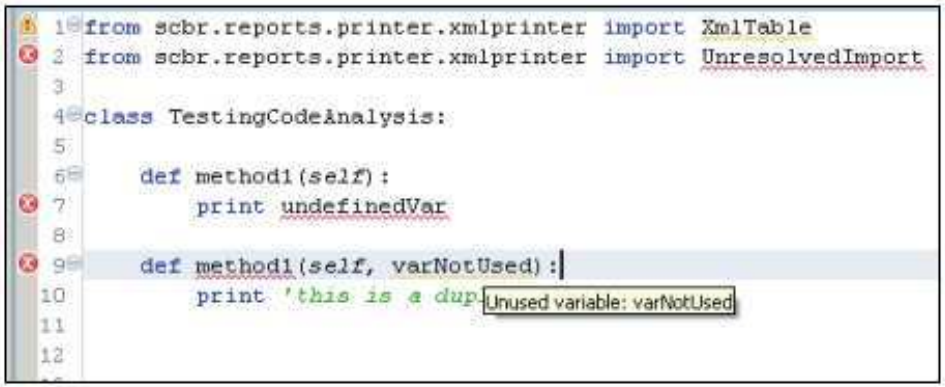
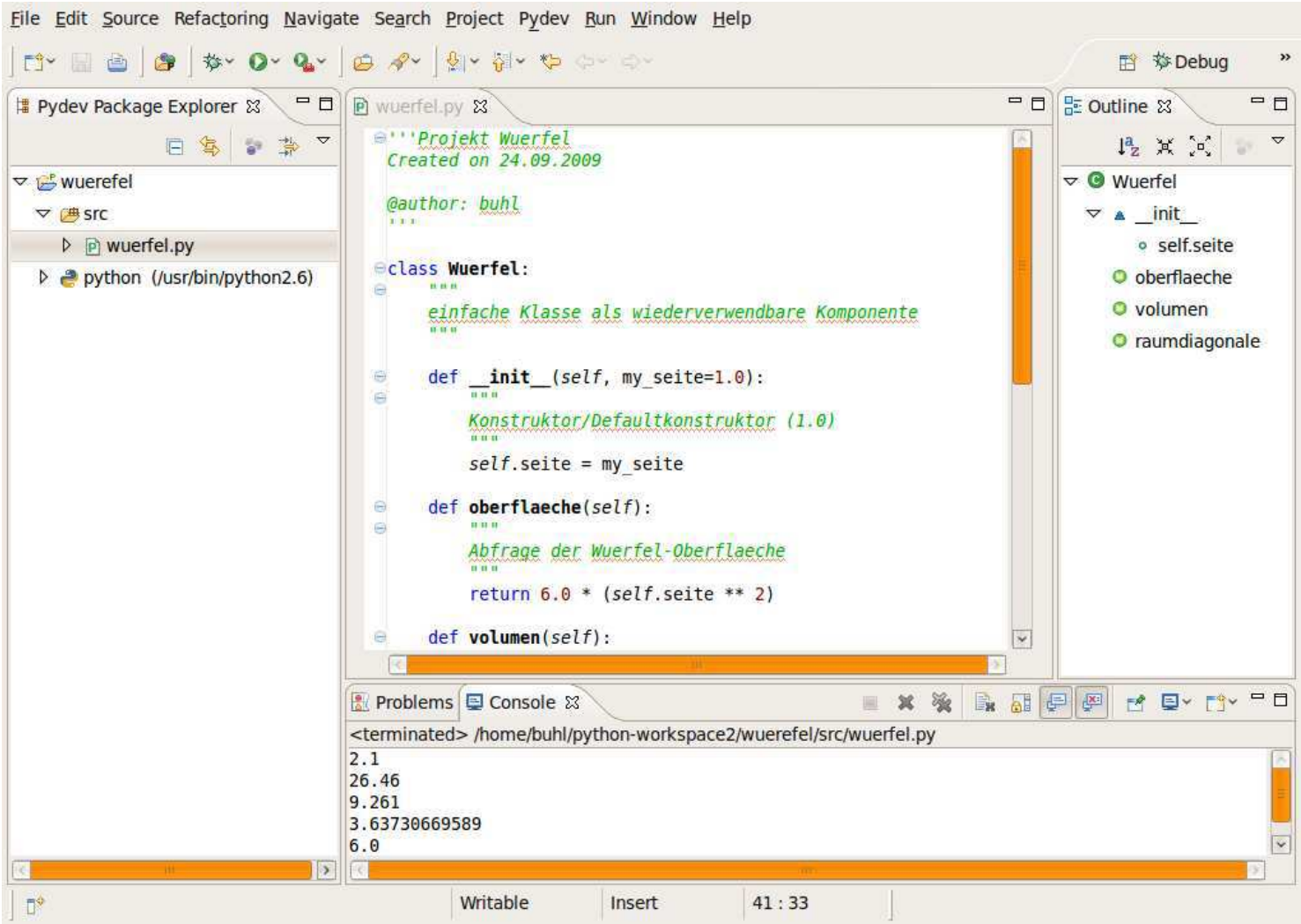
Einfache Wiederholbarkeit von Programmläufen

... durch den Einsatz einer integrierten Entwicklungsumgebung

(für Python zum Beispiel
http://pydev.org/manual.html: Pydev



)


File Edit Source Refactoring Navigate Search Project Pydev Run Window Help
Debug
Pydev Package Explorer
wuerefel
src
wuerefel.py
python (/usr/bin/python2.6)
wuerefel.py
'''Projekt Wuerfel
Created on 24.09.2009
@author: buhl
'''
class Wuerfel:
"""
einfache Klasse als wiederverwendbare Komponente
"""
def __init__(self, my_seite=1.0):
"""
Konstruktor/Defaultkonstruktor (1.0)
"""
self.seite = my_seite
def oberflaeche(self):
"""
Abfrage der Wuerfel-Oberflaeche
"""
return 6.0 * (self.seite ** 2)
def volumen(self):
Outline
Wuerfel
__init__
self.seite
oberflaeche
volumen
raumdiagonale
Problems
Console
<terminated> /home/buhl/python-workspace2/wuerefel/src/wuerfel.py
2.1
26.46
9.261
3.63730669589
6.0
Writable
Insert
41 : 33

Jede Klasse mit einem Testrahmenprogramm I

Jede Klasse (zum Beispiel `wuerfel` in `wuerfel.py`) kann mit einem eigenen kleinen Testrahmenprogramm, eingeleitet durch `if __name__ == '__main__':`, angereichert werden:

```
'''Projekt Wuerfel
Created on 24.09.2009
@author: buhl
'''

class Wuerfel:
    """
    einfache Klasse als wiederverwendbare Komponente
    """

    def __init__(self, my_seite=1.0):
```

Jede Klasse mit einem Testrahmenprogramm II

```
        """
        Konstruktor/Defaultkonstruktor (1.0)
        """
        self.seite = my_seite

#...
    def raumdiagonale(self):
        """
        Abfrage der Raumdiagonalen
        """
        return (3.0 ** 0.5) * self.seite

if __name__ == '__main__':
        w = Wuerfel(2.1)
```

Jede Klasse mit einem Testrahmenprogramm III

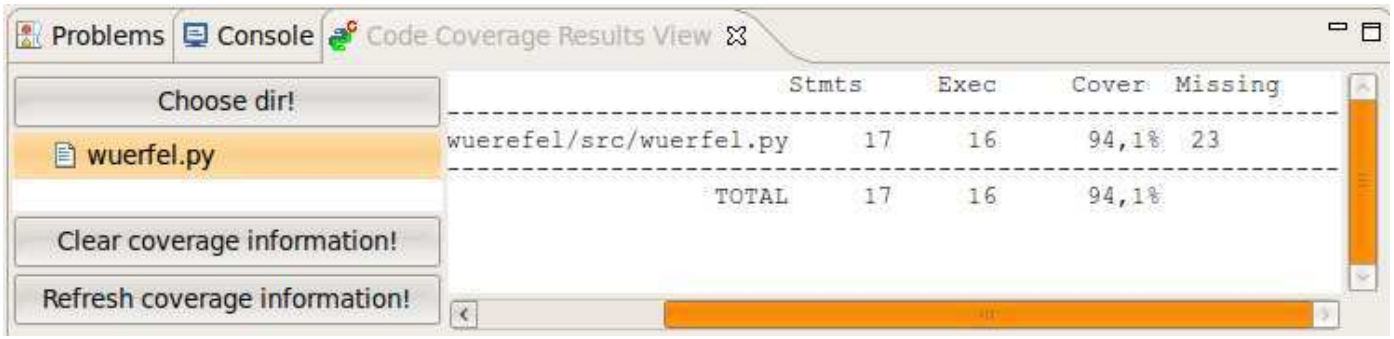
```
print(w.seite)
print(w.oberflaeche())
print(w.volumen())
print(w.raumdiagonale())
w = Wuerfel()
print(w.oberflaeche())
```

Dann reicht ein einfacher Klick auf den Run-Knopf  : alle Anweisungen werden ausgeführt und die Ergebnisse jedesmal erneut im `Console`-Fenster angezeigt.
Beim Import von `wuerfel.py` wird dieses Testrahmenprogramm jedoch nicht ausgeführt sondern nur die Klasse `wuerfel` verfügbar gemacht.

Weitere Leckerbissen von Pydev

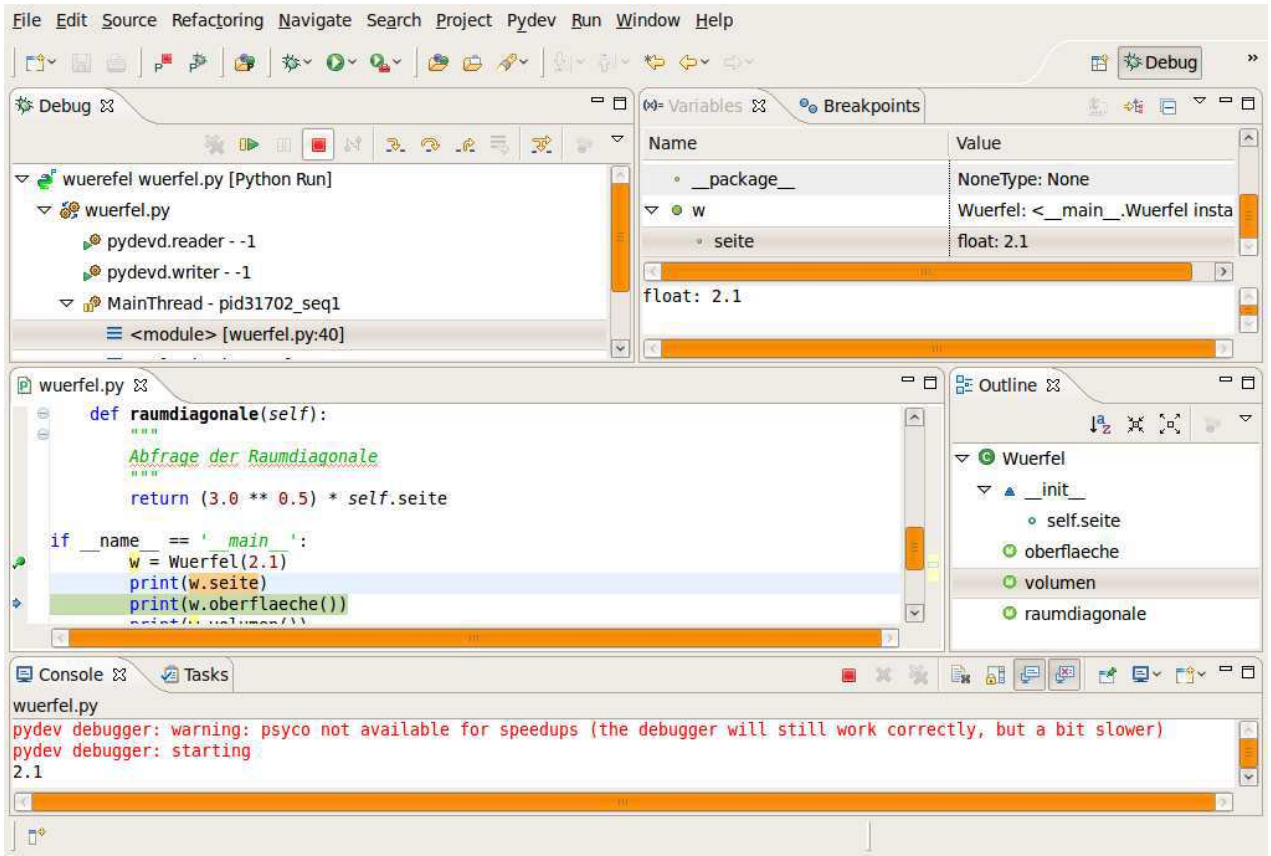
Darüber hinaus bietet Pydev zusätzliche Schmankerln:

- Integration des Python Code-Abdeckungstools:



zum Auffinden vergessener Tests.

- Python-Debugging:



zum zeilenweise Durchlaufen des Komponentencodes unter gleichzeitiger Beobachtung der Änderung der Attribut-Werte, Analyse von Fehlerabbrüchen, ...

Komponententests

- Ein erster Schritt zum häufigeren Einsatz wiederverwendbarer Softwarekomponenten ist eine geeignete, dem (programmierenden) Benutzer leicht (online) zugängliche und übersichtliche Dokumentation. pydoc liefert eine solche als html-Datei:

wuerfel — index /home/buhl/python-workspace2/wuerefel/src/wuerfel.py

Projekt Wuerfel
Created on 24.09.2009

@author: buhl

Classes

Wuerfel

class **Wuerfel**
einfache Klasse als wiederverwendbare Komponente

Methods defined here:

__init__(self, my_seite=1.0)
Konstruktor/Defaultkonstruktor (1.0)

oberflaeche(self)
Abfrage der Wuerfel-Oberflaeche

raumdiagonale(self)
Abfrage der Raumdiagonale

volumen(self)
Abfrage des Wuerfel-Volumens

- epydoc liefert eine schönere Dokumentation (ein ganzer html-Dateibaum oder eine pdf-Datei):

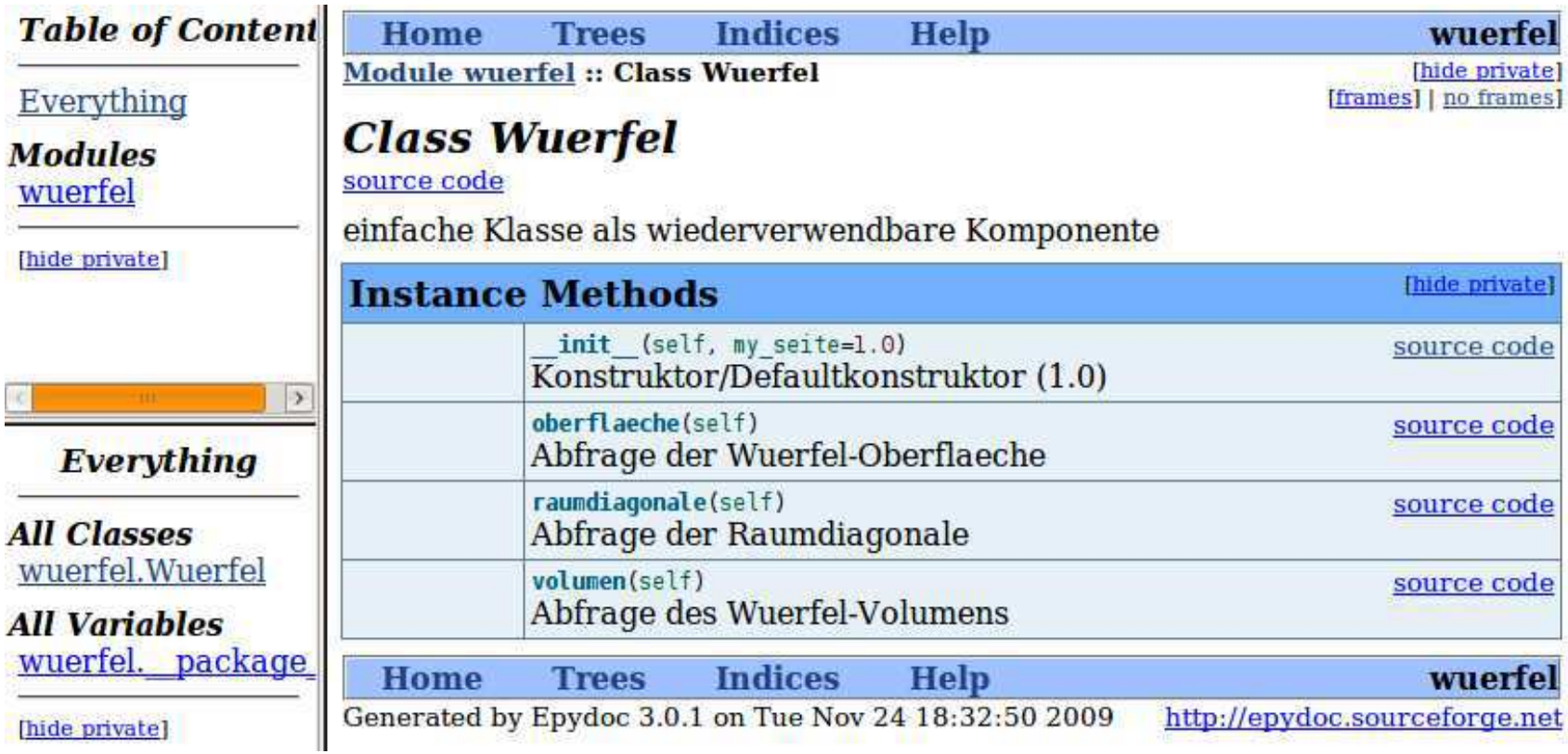


Table of Content

Everything

Modules
wuerfel

[hide private]

Everything

All Classes
wuerfel.Wuerfel

All Variables
wuerfel.__package__

[hide private]

Home Trees Indices Help wuerfel

Module wuerfel :: Class Wuerfel

[hide private]
[frames] | no frames]

Class Wuerfel

source code

einfache Klasse als wiederverwendbare Komponente

Instance Methods [hide private]

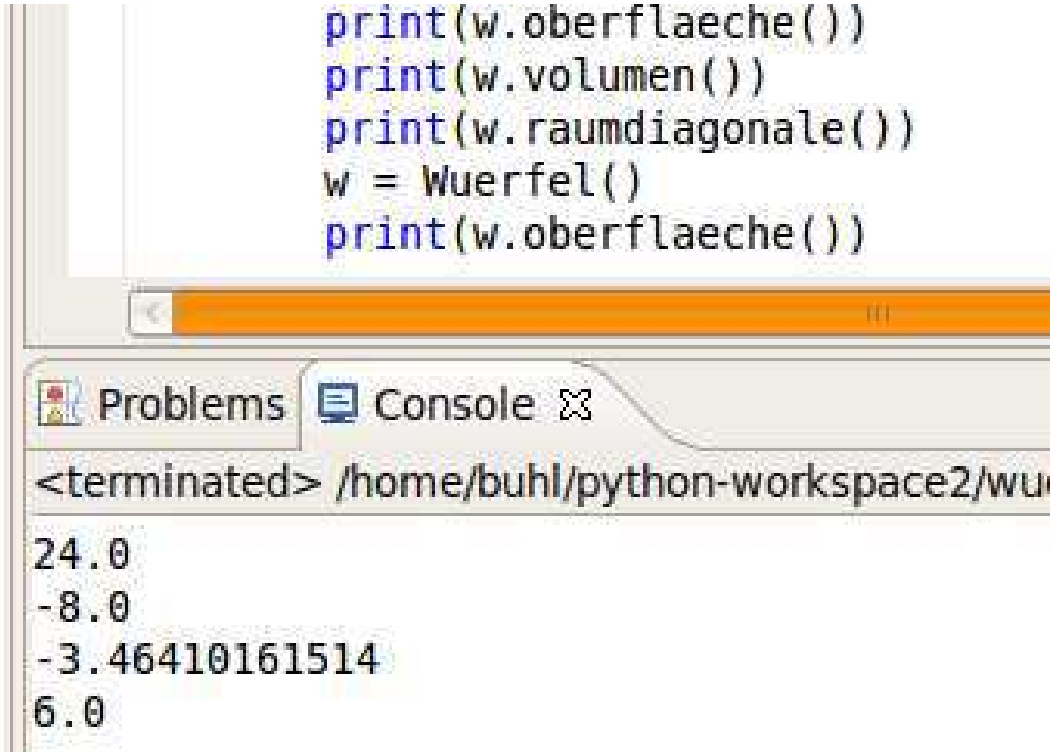
	__init__(self, my_seite=1.0) Konstruktor/Defaultkonstruktor (1.0)	source code
	oberflaeche(self) Abfrage der Wuerfel-Oberflaeche	source code
	raumdiagonale(self) Abfrage der Raumdiagonale	source code
	volumen(self) Abfrage des Wuerfel-Volumens	source code

Home Trees Indices Help wuerfel

Generated by Epydoc 3.0.1 on Tue Nov 24 18:32:50 2009 http://epydoc.sourceforge.net

- Ist unsere Dokumentation der Klasse `wuerfel` so kurz und genau wie möglich?

- Ist unsere Dokumentation der Klasse `wuerfel` so kurz und genau wie möglich?



Ergänzen wir sie unter Zuhilfenahme der in epydoc verfügbaren erweiterten Tags, so erhalten wir:

Table of Contents

Everything

Modules
wuerfel

[hide private]

Everything

All Classes
wuerfel.Wuerfel

All Variables
wuerfel.__package__

[show private]

Home Trees Indices Help **wuerfel**

Module wuerfel :: Class Wuerfel

[show private]
[frames] | no frames]

Class Wuerfel

source code

einfache Klasse als wiederverwendbare Komponente

Instance Methods [show private]

	__init__(self, my_seite=1.0) Konstruktor/Defaultkonstruktor (1.0)	source code
number	oberflaeche(self) Abfrage der Wuerfel-Oberflaeche	source code
number	raumdiagonale(self) Abfrage der Raumdiagonale	source code
number	volumen(self) Abfrage des Wuerfel-Volumens	source code

Instance Variables [show private]

	ivar seite

Method Details [show private]

__init__(self, my_seite=1.0) source code
(Constructor)

Konstruktor/Defaultkonstruktor (1.0)

Parameters:
- my_seite (number) - Seitenlaenge des zu erzeugenden Wuerfels.

Precondition: my_seite >= 0.0

oberflaeche(self) source code

Abfrage der Wuerfel-Oberflaeche

Returns: number
Oberflaeche des Wuerfels

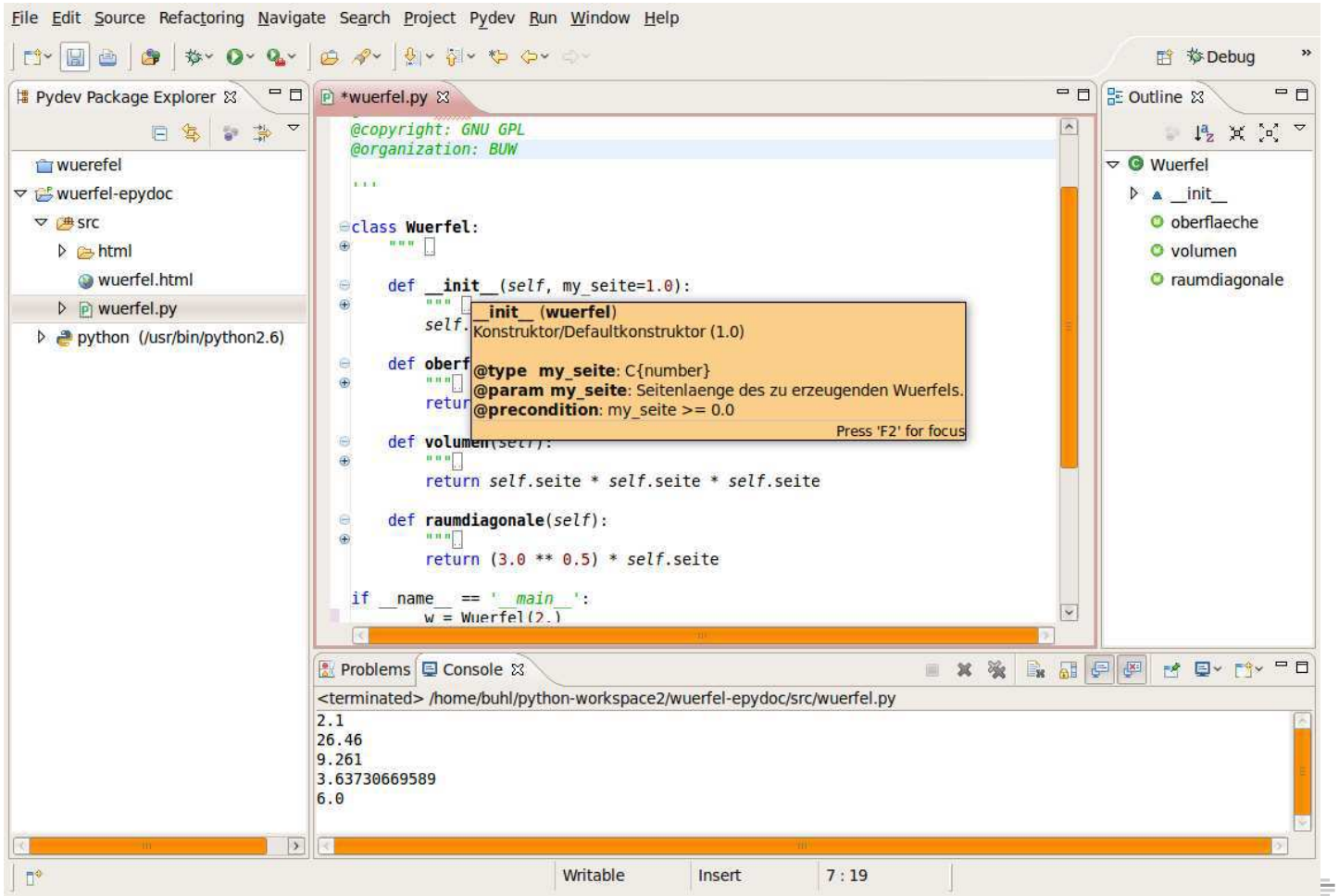
Nötige Code-Ergänzungen:

```
'''Projekt Wuerfel

@version: 1.0
@date: 24.09.2009
@author: buhl
@copyright: GNU GPL
@organization: BUW

'''

class Wuerfel:
    """
    einfache Klasse als wiederverwendbare Komponente
```

```
    @ivar seite: Seitenlaenge
    """

    def __init__(self, my_seite=1.0):
        """
        Konstruktor/Defaultkonstruktor (1.0)

        @type  my_seite: C{number}
        @param my_seite: Seitenlaenge des zu erzeugenden
        @precondition: my_seite >= 0.0
        """
        self.seite = my_seite

    def oberflaeche(self):
        """
        Abfrage der Wuerfel-Oberflaeche
```

```
        @rtype: number
        @return: Oberflaeche des Wuerfels
        """
        return 6.0 * (self.seite ** 2)

# ...
```

Auch die pydev-Entwicklungsumgebung zeigt die erweiterte Hilfestellung beim Programmieren an:



- Zur Dokumentation gehören Beispiele. Diese kann man in Python in den Dokumentationsstrings folgendermaßen angeben:

```
class Wuerfel:
    """
    einfache Klasse als wiederverwendbare Komponente

    @ivar seite: Seitenlaenge

    >>> w = Wuerfel(2.1)
    >>> print(w.seite)
    >>> print(w.raumdiagonale())

    """

# ...
```

- Ersetzt man dann das Testrahmenprogramm durch die Zeilen

```
if __name__ == '__main__':
    import doctest
    doctest.testmod()
```

so werden beim Aufruf von `wuerfel.py` alle Dokumentationsstrings (des Moduls, aller Klassen, Methoden und Funktionen) nach interaktiven Beispielen durchsucht und getestet, ob diese immer noch funtionieren:

```
**********************************************************************
File "/home/buhl/python-workspace2/wuerfel-doctest/src/w
Failed example:
    print(w.seite)
Expected nothing
Got:
    2.1
**********************************************************************
File "/home/buhl/python-workspace2/wuerfel-doctest/src/w
Failed example:
    print(w.raumdiagonale())
Expected nothing
Got:
    3.63730669589
**********************************************************************
```

```
1 items had failures:
   2 of   3 in __main__.Wuerfel
***Test Failed*** 2 failures.
```

Die Dokumentation sagte ja, dass

```
    >>> w = Wuerfel(2.1)
    >>> print(w.seite)
    >>> print(w.raumdiagonale())
```

keinerlei Output produziert wird.

- Richtig wäre gewesen:

```
class Wuerfel:
    """
    einfache Klasse als wiederverwendbare Komponente

    @ivar seite: Seitenlaenge

    >>> w = Wuerfel(2.1)
    >>> print(w.seite)
    2.1
    >>> print(w.raumdiagonale())
    3.63730669589

    """
```

Jetzt erfolgt keine Ausgabe, da alles in Ordnung ist.

- Wer stattdessen nach der Durchführung der interaktiven Tests sehen möchte, wie viele Tests durchgeführt wurden, hat lediglich `doctest.testmod()` zu `print doctest.testmod()` abzuändern:

```
TestResults(failed=0, attempted=3)
```

- Wer stattdessen nach der Durchführung der interaktiven Tests sehen möchte, wie viele Tests durchgeführt wurden, hat lediglich `doctest.testmod()` zu `print doctest.testmod()` abzuändern:

```
TestResults(failed=0, attempted=3)
```

- **Bei jedem Start von `wuerfel.py` werden alle interaktiven Tests erneut durchgeführt!** (Regressionstests)

- Abschließend eine Änderung des Konstruktors:

```
def __init__(self, my_seite=1.0):
    """
    Konstruktor/Defaultkonstruktor (1.0)

    @type  my_seite: C{number}
    @param my_seite: Seitenlaenge des zu erzeuge
    @precondition: my_seite >= 0.0
    @raise ValueError: falls my_seite < 0.0
    """
    if my_seite < 0:
        raise ValueError
    self.seite = my_seite
```

(Nur die Änderung der Dokumentation ist natürlich nicht ausreichend gewesen!)

und ein zugehöriger interaktiver Testfall:

```
class Wuerfel:
    """
    einfache Klasse als wiederverwendbare Komponente
    @ivar seite: Seitenlaenge

    >>> w = Wuerfel(2.1)
    >>> print(w.seite)
    2.1
    >>> print(w.raumdiagonale())
    3.63730669589
    >>> w2 = Wuerfel(-2.0)
    Traceback (most recent call last):
    ...
    ValueError
    """
```

Verträge zwischen Nutzer und Anbieter von Klassen

Spezifikation durch Verträge

SdV	VERPFLICHTUNGEN	NUTZEN
Nutzer der Klasse	Aufruf von Klassenmethoden nur bei erfüllter Vorbedingung	kann auf erfüllte Nachbedingung vertrauen
Anbieter der Klasse	garantiert die Nachbedingung	kann von erfüllter Vorbedingung ausgehen

Tabelle: Verpflichtungen/Nutzen von Verträgen zwischen Anbieter und Nutzer

Sollte es zu einem Softwarefehler kommen, ist der Verantwortliche sofort klar, sobald man festgestellt hat, ob die Vorbedingung oder die Nachbedingung (bei gültiger Vorbedingung) verletzt wurde.

Klasseninvariante, Methodenvor- und -nachbedingungen

- Eine Klasseninvariante ist zur gesamten Lebenszeit der Klasseninstanz `true`.

  Das Laufzeitsystem überprüft am Ende des Konstruktor-Aufrufs, am Beginn des Destruktor-Aufrufs und am Beginn und Ende jedes anderen Methoden-Aufrufs die Gültigkeit der Klasseninvariante. Es überprüft, ob jede Methode bei anfänglich gültiger Invariante und Vorbedingung zur gültigen Nachbedingung führt.

  Ein Beispiel für die Klasse `wuerfel` unter Benutzung des Python-Moduls Contract:

Klasseninvariante, Methodenvor- und -nachbedingungen

- Eine Klasseninvariante ist zur gesamten Lebenszeit der Klasseninstanz `true`.

  Das Laufzeitsystem überprüft am Ende des Konstruktor-Aufrufs, am Beginn des Destruktor-Aufrufs und am Beginn und Ende jedes anderen Methoden-Aufrufs die Gültigkeit der Klasseninvariante. Es überprüft, ob jede Methode bei anfänglich gültiger Invariante und Vorbedingung zur gültigen Nachbedingung führt.

  Ein Beispiel für die Klasse `wuerfel` unter Benutzung des Python-Moduls Contract:

Klasseninvariante, Methodenvor- und -nachbedingungen

- Eine Klasseninvariante ist zur gesamten Lebenszeit der Klasseninstanz `true`.

  Das Laufzeitsystem überprüft am Ende des Konstruktor-Aufrufs, am Beginn des Destruktor-Aufrufs und am Beginn und Ende jedes anderen Methoden-Aufrufs die Gültigkeit der Klasseninvariante. Es überprüft, ob jede Methode bei anfänglich gültiger Invariante und Vorbedingung zur gültigen Nachbedingung führt.

  Ein Beispiel für die Klasse `wuerfel` unter Benutzung des Python-Moduls Contract:

```
'''Projekt Wuerfel

@version: 1.0

'''

import sys

class Wuerfel:
    """
    einfache Klasse als wiederverwendbare Komponente

    @ivar seite: Seitenlaenge

    >>> w = Wuerfel(2.1)
```

```
>>> print(w.seite)
2.1
>>> print(w.raumdiagonale())
3.63730669589
>>> w2 = Wuerfel(-2.0)
Traceback (most recent call last):
...
PreconditionViolationError: ('__main__.Wuerfel.__ini

inv:
    hasattr(self, 'seite')
    isinstance(self.seite, float)
    self.seite >= 0.0
"""

def __init__(self, my_seite=1.0):
```

```
        """
        Konstruktor/Defaultkonstruktor (1.0)

        @type  my_seite: C{number}
        @param my_seite: Seitenlaenge des zu erzeugenden
        @precondition: my_seite >= 0.0

        pre:
            isinstance(my_seite, float
              ) or isinstance(my_seite, int)
            my_seite >= 0.0
        post:
            __return__ == None
            self.seite == my_seite
    """
        self.seite = float(my_seite)
```

```
def oberflaeche(self):
    """
    Abfrage der Wuerfel-Oberflaeche

    @rtype: number
    @return: Oberflaeche des Wuerfels

    post[self.seite]:
        self.seite == __old__.self.seite
        __return__ == 6.0 * (
            self.seite * self.seite)

    """
    return 6.0 * (self.seite ** 2)
```

```
# ...
import contract
contract.checkmod(__name__)

if __name__ == '__main__':
        import doctest
        print doctest.testmod()

        w = Wuerfel(2.0)
        print w.seite
        print w.oberflaeche()
        print w.volumen()
        print w.raumdiagonale()
```

Was geschieht bei Vertragsverletzungen?

- Es wird eine Ausnahmebedingung ausgelöst.

Was geschieht bei Vertragsverletzungen?

- Es wird eine Ausnahmebedingung ausgelöst.
- bei Verletzung einer Vorbedingung:

```
Traceback (most recent call last):
  File "/home/buhl/python-workspace2/wuerfel-contract
    w = Wuerfel(-2.0)
...
contract.PreconditionViolationError:
          ('__main__.Wuerfel.__init__', 11)
```

- bei Verletzung einer Nachbedingung:

```
Traceback (most recent call last):
  File "/home/buhl/python-workspace2/wuerfel-contract
    print w.oberflaeche()
...
contract.PostconditionViolationError:
          ('__main__.Wuerfel.oberflaeche', 8)
```

- Der Nutzer reagiert i.A. in der folgenden Form auf Ausnahmebedingungen:

```
try:
    w = Wuerfel(x-y*4.0)
except PreconditionViolationError:
    print "Objekt der Klasse Wuerfel nicht er
    # zur Haupteingabeschleife oder:
    sys.exit(1)
# normale Programmfortsetzung
```

Verantwortlichkeitszuweisung/Invarianten

- Eine Invariante hätte das Therac-25-Problem vermieden (`BestrahlungEin()` hätte nicht aktiviert werden können)

Verantwortlichkeitszuweisung/Invarianten

- Eine Invariante hätte das Therac-25-Problem vermieden (`BestrahlungEin()` hätte nicht aktiviert werden können)
- Methodenverträge hätten während der Testphase viele Probleme aufgedeckt (zum Beispiel in java.awt)

Verantwortlichkeitszuweisung/Invarianten

- Eine Invariante hätte das Therac-25-Problem vermieden (`BestrahlungEin()` hätte nicht aktiviert werden können)
- Methodenverträge hätten während der Testphase viele Probleme aufgedeckt (zum Beispiel in java.awt)
- Im Gegensatz zum Komponententest werden Verträge bei jedem Test und meist auch bei jedem Produktionslauf überprüft

Verantwortlichkeitszuweisung/Invarianten

- Eine Invariante hätte das Therac-25-Problem vermieden (`BestrahlungEin()` hätte nicht aktiviert werden können)
- Methodenverträge hätten während der Testphase viele Probleme aufgedeckt (zum Beispiel in java.awt)
- Im Gegensatz zum Komponententest werden Verträge bei jedem Test und meist auch bei jedem Produktionslauf überprüft
- Selbst wenn bei Produktionsläufen die Nachbedingungen nicht mehr überprüft werden, so doch zumindest die Vorbedingungen

Verantwortlichkeitszuweisung/Invarianten

- Eine Invariante hätte das Therac-25-Problem vermieden (`BestrahlungEin()` hätte nicht aktiviert werden können)
- Methodenverträge hätten während der Testphase viele Probleme aufgedeckt (zum Beispiel in java.awt)
- Im Gegensatz zum Komponententest werden Verträge bei jedem Test und meist auch bei jedem Produktionslauf überprüft
- Selbst wenn bei Produktionsläufen die Nachbedingungen nicht mehr überprüft werden, so doch zumindest die Vorbedingungen
- Verträge enthalten durch die Nachbedingung ein eingebautes Test-Orakel

Ein Vertrag für eine globale Funktion

- Wie sollte ein Vertrag für die folgende Funktion zur Berechnung der m-ten Wurzel nach dem Newton-Verfahren aussehen?

```
def sqrtm(n,m):
    """m-te Wurzel
    """
    approx = n/m
    better = approx*(m - 1 + n/approx**m)/m
    while better != approx:
        approx = better
        better = approx*(m - 1 + n/approx**m)/m
    return approx
```

Vergleiche etwa „How to Think Like a Computer Scientist“

- Erste interaktive Testversuche liefern:

```
>>> sqrtm(512.0, 9)
2.0
>>> sqrtm(512, 9)
2
>>> sqrtm(0.000001, 6)
0.1
>>> sqrtm(-1, 2)
```

eine Endlosschleife (bitte in Pydev abbrechen(!), da sonst bei mehrfachem Start unbenutzbare Maschine zurückbleibt oder sogar die CPU überhitzt wird)

- Weiter:

```
>>> sqrtm(0.0, 2)
Traceback (most recent call last):
...
ZeroDivisionError: float division
```

- und:

```
>>> sqrtm('a',2)
Traceback (most recent call last):
...
TypeError: unsupported operand type(s) for /: 'str' a
```

- Weiter:

```
>>> sqrtm(0.0, 2)
Traceback (most recent call last):
...
ZeroDivisionError: float division
```

- und:

```
>>> sqrtm('a',2)
Traceback (most recent call last):
...
TypeError: unsupported operand type(s) for /: 'str' a
```

- Verbesserung der Implementierung:

```
def sqrtm(n,m):
# ...
    n = float(n)
    if n == 0:
        return n
    approx = n/m
    better = approx*(m - 1 + n/approx**m)/m
#...
```

- Erster Vertragsentwurf:

```
def sqrtm(n,m):
    """m-te Wurzel
    pre:
        isinstance(n, float) or isinstance(n, int)
        n >= 0.0
        isinstance(m, int)
        m > 1
    post:
        __return__  ** m == n
    """
# ...
```

- mit dem Ergebnis:

```
Traceback (most recent call last):
  File "/home/buhl/python-workspace2/wuerfel-contrac
    print sqrtm1(0.000001, 6)
...
contract.PostconditionViolationError: ('__main__.sqr
```

Warum?

- Untersuchung:

```
>>> sqrtm(0.000001, 6)
0.1
>>> repr(sqrtm(0.000001, 6))
0.09999999999999992
>>> repr(sqrtm(0.000001, 6)-0.1))
-1.3877787807814457e-17
```

- mit dem Ergebnis:

```
Traceback (most recent call last):
  File "/home/buhl/python-workspace2/wuerfel-contrac
    print sqrtm1(0.000001, 6)
...
contract.PostconditionViolationError: ('__main__.sqr
```

Warum?

- Untersuchung:

```
>>> sqrtm(0.000001, 6)
0.1
>>> repr(sqrtm(0.000001, 6))
0.09999999999999992
>>> repr(sqrtm(0.000001, 6)-0.1))
-1.3877787807814457e-17
```

- Zweiter Vertragsentwurf:

```
def sqrtm(n,m):
    """m-te Wurzel
    pre:
      isinstance(n, float) or isinstance(n, int)
      n >= 0.0
      isinstance(m, int)
      m > 1
    post:
      approximately_equal_to(__return__  ** m,
                             n,
                             1.0)
    """
# ...
```

```
approximately_equal_to(arg1, arg2, rel_distance):
    """ controlled relative distance.
    """
    return ( abs(arg1 - arg2) <=
             sys.float_info.epsilon *
             rel_distance *
             max(abs(arg1), abs(arg2)) )
```

- mit dem neuen Ergebnis:

```
Traceback (most recent call last):
...
contract.PostconditionViolationError: ('__main__.sqr
```

Warum?

- Untersuchung:

```
>>> sys.float_info.epsilon
2.22044604925e-16
>>> abs((sqrtm(0.000001, 6)**6-0.000001))/0.000001
4.23516473627e-16
```

Da hat der Anbieter den Mund mit der erreichbaren Genauigkeit wohl zu voll genommen!

- mit dem neuen Ergebnis:

```
Traceback (most recent call last):
...
contract.PostconditionViolationError: ('__main__.sqr
```

Warum?

- Untersuchung:

```
>>> sys.float_info.epsilon
2.22044604925e-16
>>> abs((sqrtm(0.000001, 6)**6-0.000001))/0.000001
4.23516473627e-16
```

Da hat der Anbieter den Mund mit der erreichbaren Genauigkeit wohl zu voll genommen!

- (vorläufig) letzter Vertragsentwurf:

```
def sqrtm(n,m):
    """m-te Wurzel
    pre:
      isinstance(n, float) or isinstance(n, int)
      n >= 0.0
      isinstance(m, int)
      m > 1
    post:
      approximately_equal_to(__return__  ** m,
                             n,
                             2.0)
    """
# ...
```