Una fugace occhiata al Test Driven Development

Roberto Bettazzoni
roberto@bettazzoni.it

Cosa è il TDD?

Cosa è il TDD?

Una pratica agile.

Pratiche agili ... cosa sono?

Pratiche agili ... cosa sono?

Sono le pratiche utilizzate nelle metodologie agili.

Metodologie agili ... cosa sono?

Metodologie agili ... cosa sono?

Sono le metodologie basate sui valori dell'Agile Manifesto

Il Manifesto dell'Alleanza Agile

Stiamo portando alla luce metodi migliori di sviluppare software facendolo in prima persona e aiutando altri a farlo. Attraverso questo tipo di lavoro siamo giunti ai seguenti valori:

Persone e interazioni più che processi e tools

Software che funziona più che una documentazione esaustiva

Collaborazione con il cliente più che negoziazione contrattuale

Rispondere al cambiamento più che seguire un piano prestabilito

Cioè, mentre c'è un valore nelle voci sulla destra,
attribuiamo un valore maggiore a quelle sulla sinistra.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas.

Metodologie e pratiche

- Pratica

Modalità per il conseguimento di uno scopo. Spesso è formulata in sequenze di passi.

Es: TDD

- Metodologia

Insieme di pratiche legate tra loro da una visione *filosofica* d'insieme.

Es: Scrum

Queste definizioni, nella loro accezione *generica*, sono fonte di numerose discussioni. Devono essere considerate come semplificazione per i nostri scopi.

Metodologie Agili

Rispondere al cambiamento

più che
seguire un piano prestabilito

Cambia il metodo di approccio al problema.

Metodologie Agili

Le metodologie 'storiche' hanno un approccio
PREDITTIVO
(elaborare un piano e seguirlo)

Le metodologie agili hanno un approccio
ADATTATIVO
(rispondere al cambiamento)

Basi del TDD

Un linguaggio di programmazione

Un sistema di Test (unit test)

Il Refactoring

Refactoring

Processo di modifica evolutiva di un sistema software in modo da non modificarne il comportamento esterno migliorandone la sua struttura

Refactoring

- modifica il codice in piccoli passi
- migliora continuamente il design
- il codice diventa «parlante»
- richiede disciplina
- la pratica rende perfetti
- richiede un buon «fiuto»

Refactoring

Alcune «puzze» comuni

1. codice duplicato
2. metodi/classi lunghe
3. lunghe liste di parametri
4. cambiamenti divergenti
5. shotgun surgery
6. codice «invidioso»
7. liste di switch/if
8. generalizzazione speculativa: «astronauti»
9. commenti!

...ed alcune soluzioni

1. estrai la parte duplicata
2. separa le competenze
3. introduci/estrai oggetti
4. separa le competenze
5. riunire la responsabilità
6. muovi/estrai metodo
7. polimorfismo
8. canc, canc, canc, canc, canc, canc, canc, canc
9. nascosto dal deodorante!

Test Driven Development (TDD)

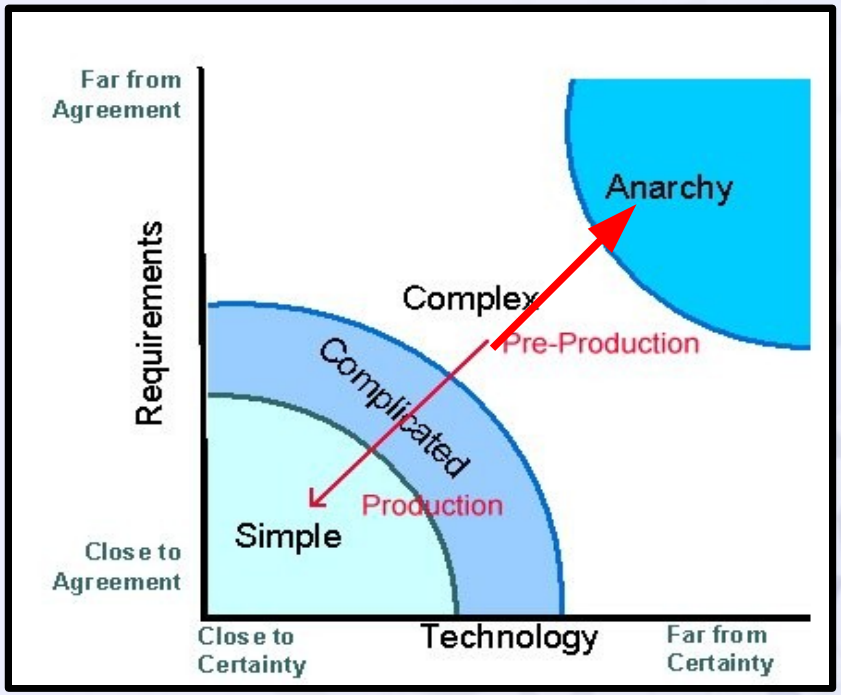
Nei processi di sviluppo software tradizionali il collaudo si fa alla fine...

Nella realtà il collaudo non viene mai fatto :(

Perchè?

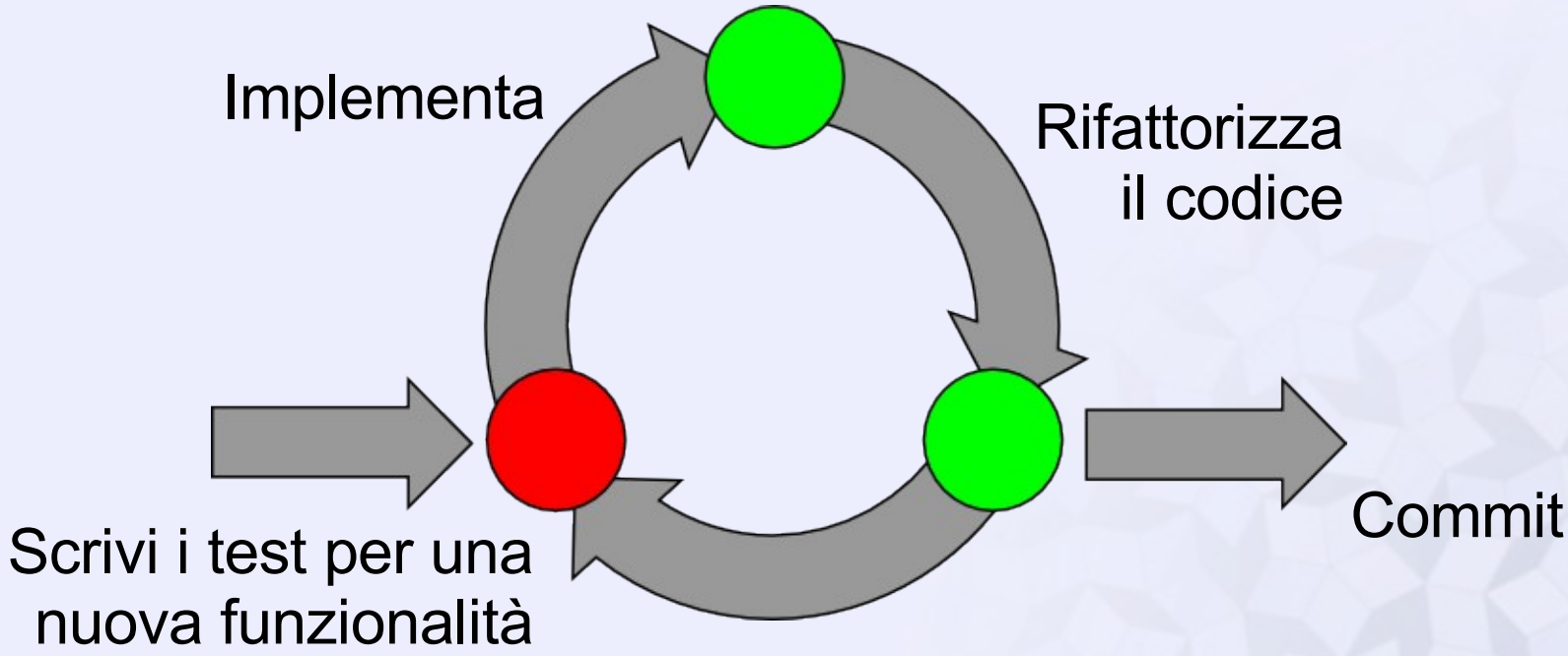
\+ stress => - test

\- test => + stress



Test Driven Development (TDD)

In TDD i test guidano lo sviluppo!



Test Driven Development (TDD)

Per un buon TDD:

- Pensa ai test prima di pensare l'implementazione
- Una «barra rossa» è meglio di uno schermo bianco
- I test devono essere semplici e leggibili
- I test devono essere isolati
- I test devono essere veloci e lanciati spesso
- Verifica prima il risultato atteso, verifica i casi limite, verifica le eccezioni
- Rifattorizza sia l'implementazione che i test

Test Driven Development (TDD)

TDD significa anche «Test Driven **Design**»

- Permette di definire l'interfaccia degli oggetti prima di buttarsi sull'implementazione
- Permette di concentrarsi sul comportamento dell'oggetto
- Permette di rifattorizzare il codice con maggiore confidenza
- Porta alla realizzazione di architetture con basso accoppiamento

Test Driven Development (TDD)

Con TDD posso anche...

- Documentare l'utilizzo dei miei componenti attraverso i test
- Evitare la regressione dei bug
- Isolare codice legacy
- Dormire un po' più tranquillo la sera dopo il rilascio

Test Driven Development (TDD)

TDD non è Unit Testing

- ...ma sono ottimi amici :)

Come testare le interfacce utente?

- Utilizzando tool ad-hoc
- Rimuovendo la logica dalle interfacce

Come testare liberandosi da risorse esterne?

- Mock

TDD può creare dipendenza! :)

Test Driven Development (TDD)

Strumenti:

- *Unit: JUnit, PyUnit, NUnit, CPPUnit, PHPUnit, ...
- DBUnit, SQLUnit, PL/SQL Unit, ...
- EasyMock, Moquer, MockEJB, ...
- Selenium, HttpUnit, Jmeter, ...
- Fit, Fitnesse, ...

...forse pure troppi

Test Driven Development (TDD)

DOMANDE?

Esempi di TDD

(Test Driven Development)

Roberto Bettazzoni
roberto@bettazzoni.it

Primo Esempio di TDD

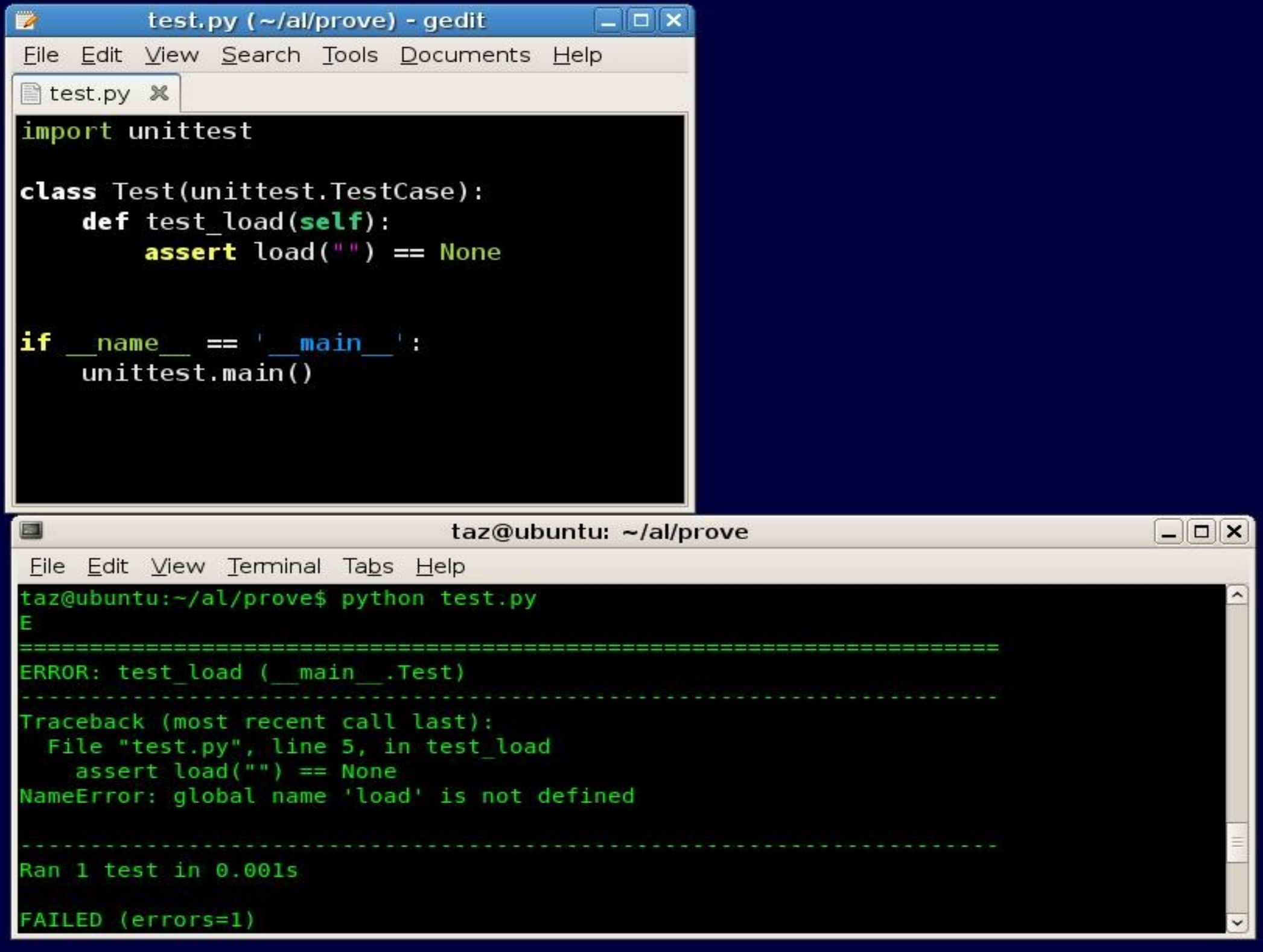
Prima User Story.

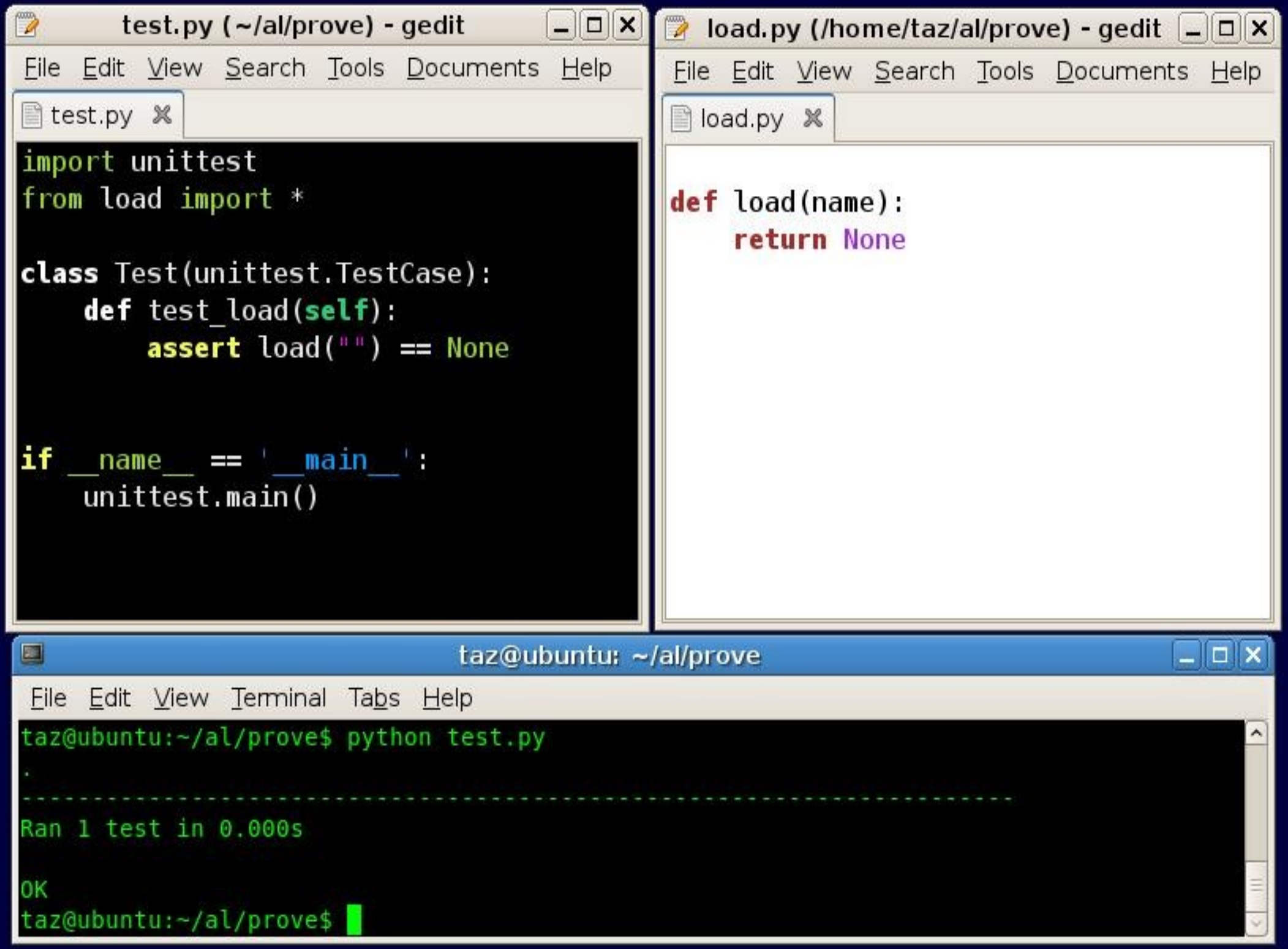
Una funzione XML-RPC che, dato il nome di un file presente sul server, ne ritorni il contenuto

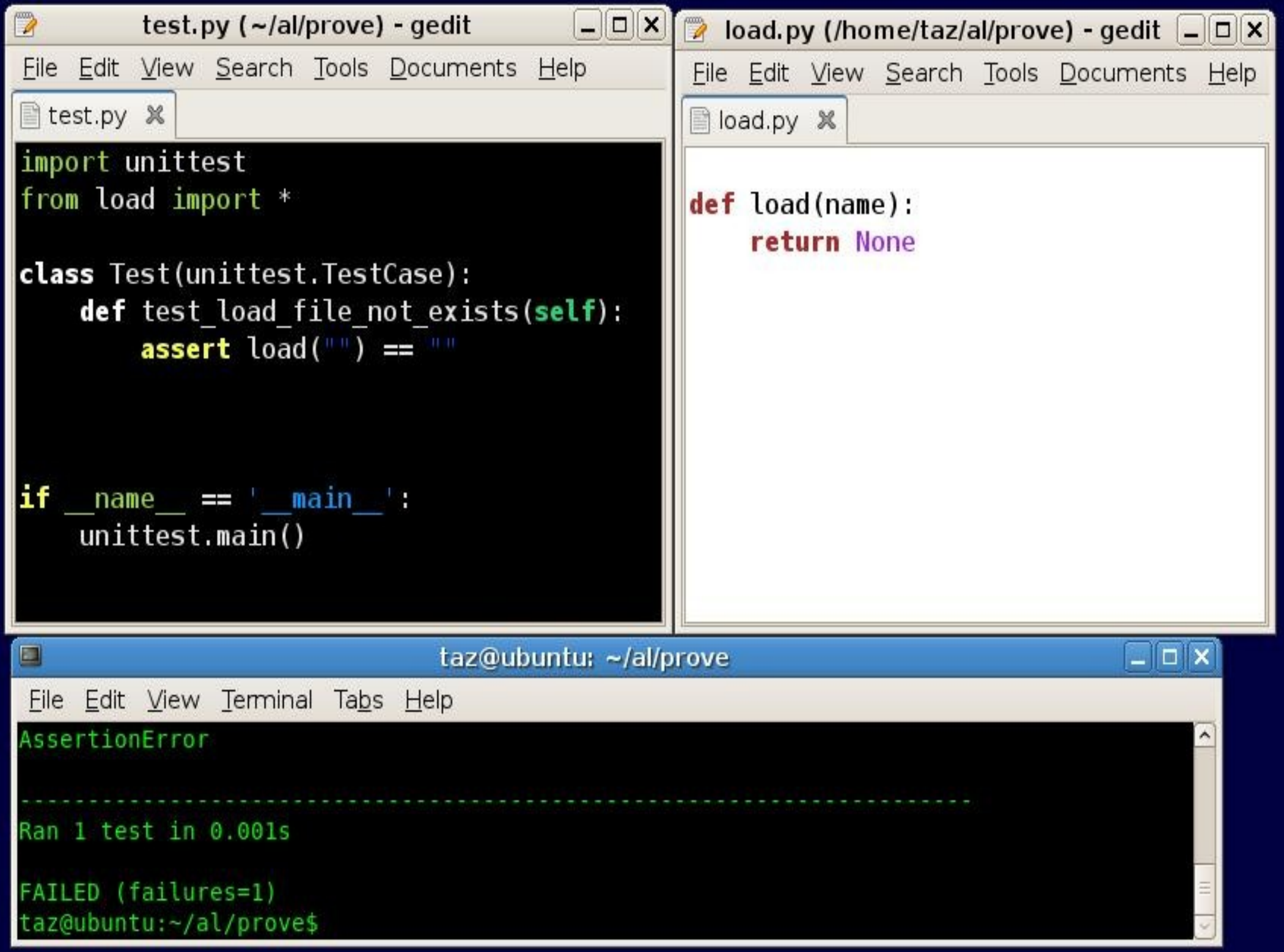
Primo Esempio di TDD

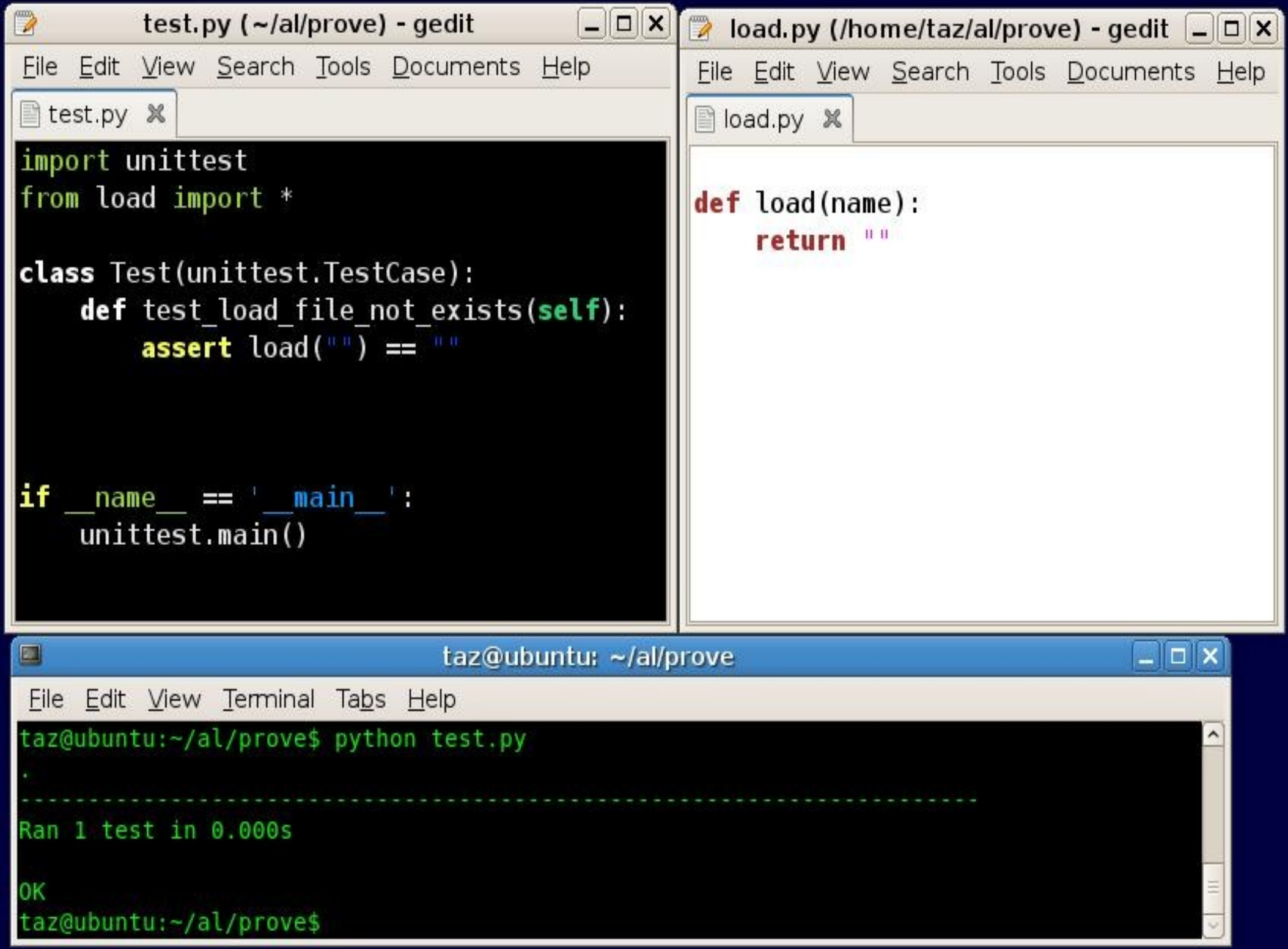
Acceptance test.

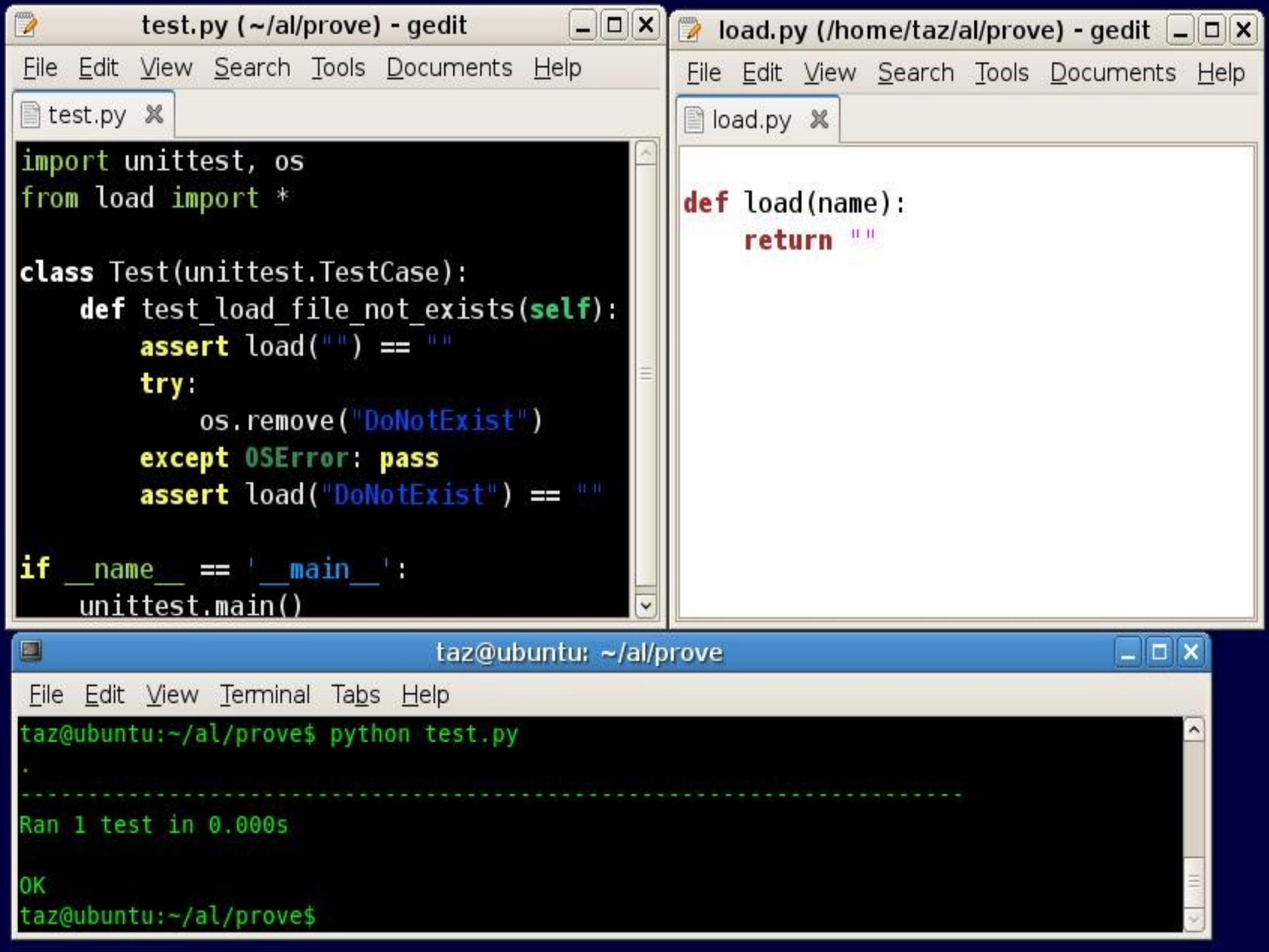
```
#!/usr/bin/env python

from xmlrpclib import ServerProxy

srv = ServerProxy('http://localhost:8000')
f = open(__file__, "rt")
try:
    assert srv.load(__file__) == f.read()
finally:
    f.close()
```

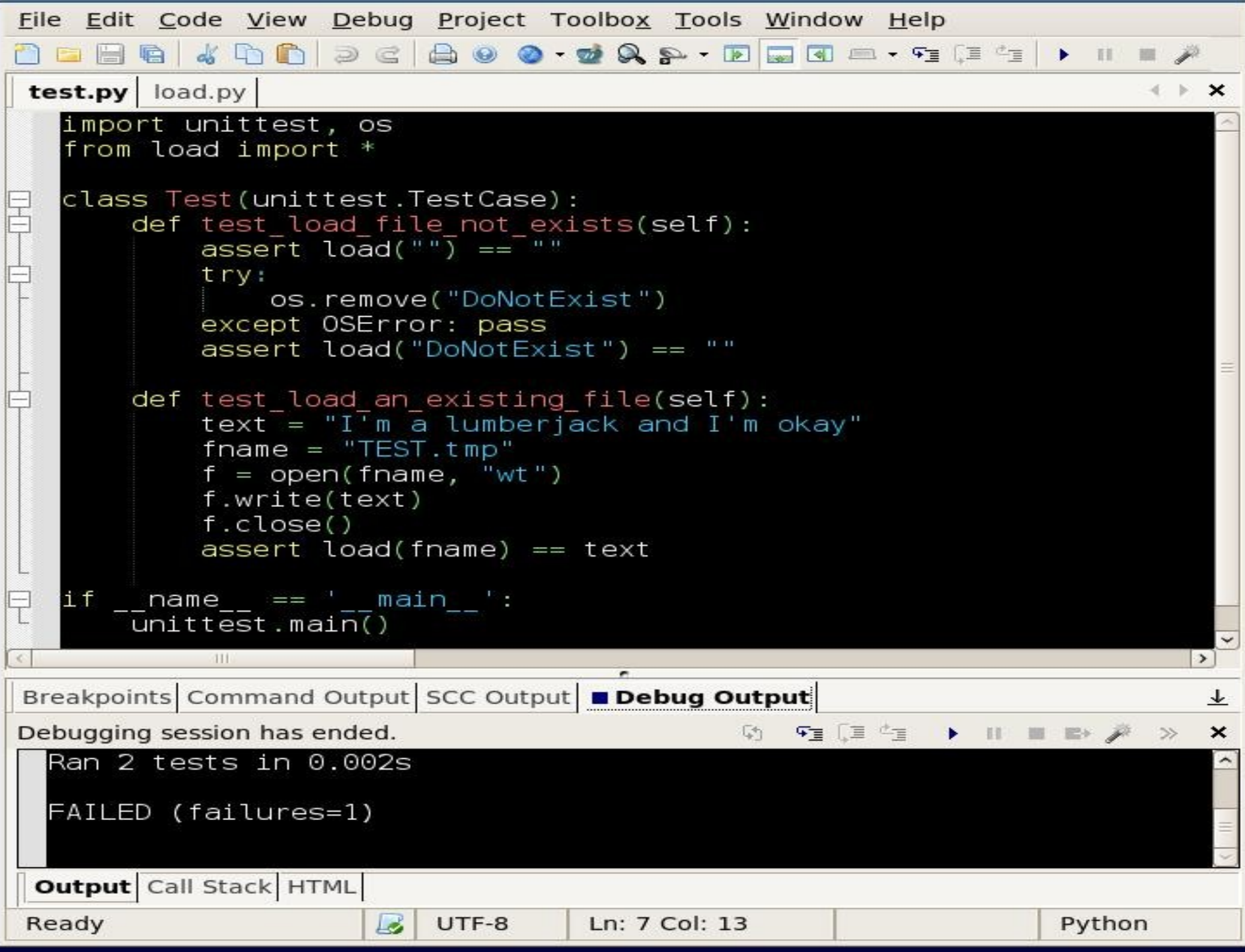

test.py (~/al/prove) - gedit
File Edit View Search Tools Documents Help
test.py
import unittest
class Test(unittest.TestCase):
def test_load(self):
assert load("") == None
if __name__ == '__main__':
unittest.main()
taz@ubuntu: ~/al/prove
File Edit View Terminal Tabs Help
taz@ubuntu:~/al/prove$ python test.py
E
ERROR: test_load (__main__.Test)
Traceback (most recent call last):
File "test.py", line 5, in test_load
assert load("") == None
NameError: global name 'load' is not defined
Ran 1 test in 0.001s
FAILED (errors=1)


test.py (~/al/prove) - gedit
File Edit View Search Tools Documents Help
test.py
import unittest
from load import *
class Test(unittest.TestCase):
def test_load(self):
assert load("") == None
if __name__ == '__main__':
unittest.main()
load.py (/home/taz/al/prove) - gedit
File Edit View Search Tools Documents Help
load.py
def load(name):
return None
taz@ubuntu: ~/al/prove
File Edit View Terminal Tabs Help
taz@ubuntu:~/al/prove$ python test.py
.
Ran 1 test in 0.000s
OK
taz@ubuntu:~/al/prove$


test.py (~/al/prove) - gedit
File Edit View Search Tools Documents Help
test.py
import unittest
from load import *
class Test(unittest.TestCase):
def test_load_file_not_exists(self):
assert load("") == ""
if __name__ == '__main__':
unittest.main()
load.py (/home/taz/al/prove) - gedit
File Edit View Search Tools Documents Help
load.py
def load(name):
return None
taz@ubuntu: ~/al/prove
File Edit View Terminal Tabs Help
AssertionError
Ran 1 test in 0.001s
FAILED (failures=1)
taz@ubuntu:~/al/prove$


test.py (~/al/prove) - gedit
File Edit View Search Tools Documents Help
test.py
import unittest
from load import *
class Test(unittest.TestCase):
def test_load_file_not_exists(self):
assert load("") == ""
if __name__ == '__main__':
unittest.main()
load.py (/home/taz/al/prove) - gedit
File Edit View Search Tools Documents Help
load.py
def load(name):
return ""
taz@ubuntu: ~/al/prove
File Edit View Terminal Tabs Help
taz@ubuntu:~/al/prove$ python test.py
.
Ran 1 test in 0.000s
OK
taz@ubuntu:~/al/prove$


test.py (~/al/prove) - gedit
File Edit View Search Tools Documents Help
test.py
import unittest, os
from load import *
class Test(unittest.TestCase):
def test_load_file_not_exists(self):
assert load("") == ""
try:
os.remove("DoNotExist")
except OSError: pass
assert load("DoNotExist") == ""
if __name__ == '__main__':
unittest.main()
load.py (/home/taz/al/prove) - gedit
File Edit View Search Tools Documents Help
load.py
def load(name):
return ""
taz@ubuntu: ~/al/prove
File Edit View Terminal Tabs Help
taz@ubuntu:~/al/prove$ python test.py
.
Ran 1 test in 0.000s
OK
taz@ubuntu:~/al/prove$


File Edit Code View Debug Project Toolbox Tools Window Help
test.py
load.py
import unittest, os
from load import *
class Test(unittest.TestCase):
def test_load_file_not_exists(self):
assert load("") == ""
try:
os.remove("DoNotExist")
except OSError: pass
assert load("DoNotExist") == ""
def test_load_an_existing_file(self):
text = "I'm a lumberjack and I'm okay"
fname = "TEST.tmp"
f = open(fname, "wt")
f.write(text)
f.close()
assert load(fname) == text
if __name__ == '__main__':
unittest.main()
Breakpoints
Command Output
SCC Output
Debug Output
Debugging session has ended.
Ran 2 tests in 0.002s
FAILED (failures=1)
Output
Call Stack
HTML
Ready
UTF-8
Ln: 7 Col: 13
Python

test.py

```
import unittest, os
from load import *

class Test(unittest.TestCase):
    def test_load_file_not_exists(self):
        assert load("") == ""
        try:
            os.remove("DoNotExist")
        except OSError: pass
        assert load("DoNotExist") == ""

    def test_load_an_existing_file(self):
        text = "I'm a lumberjack and I'm okay"
        fname = "TEST.tmp"
        f = open(fname, "wt")
        f.write(text)
        f.close()
        assert load(fname) == text
        os.remove(fname)

if __name__ == '__main__':
    unittest.main()
```

load.py

```
def load(name):
    try:
        f = open(name, "rt")
        try:
            return f.read()
        finally:
            f.close()
    except IOError:
        return ""
```

test.py

```
import unittest, os, pyprocess, xmlrpclib
from load import *

def createFile(fname, text):
    f = open(fname, "wt")
    f.write(text)
    f.close()

class Test(unittest.TestCase):
    def test_load_file_not_exists(self):        ...
    def test_load_an_existing_file(self):       ...

class TestServer(unittest.TestCase):
    def test(self):
        proc = pyprocess.PyProcess("load.py")
        proc.start()
        createFile("other", "text")
        try:
            rpc = xmlrpclib.ServerProxy('http://localhost:8000')
            assert rpc.load("other") == "text"
        finally:
            proc.kill()
            os.remove("other")
```

load.py

```
import os, SimpleXMLRPCServer

def load(name):
    try:
        f = open(name, "rt")
        try:
            return f.read()
        finally:
            f.close()
    except IOError:
        return ""

if __name__ == '__main__':
    srv = SimpleXMLRPCServer.SimpleXMLRPCServer(("",8000))
    srv.register_function(load)
    srv.serve_forever()
```

Primo Esempio di TDD

Eseguendo il file “load.py”
il test di accettazione gira senza errori.

End of Job (?)