Testing software

Computer & Information Sciences

W. H. Bell

Overview

- Types of tests.
- Design for testing.
- Automated testing.
- Unit tests.
- Summary.

Types of tests

- Unit tests – Implemented by developers to test a unit.
- Isolate program units, providing input data and state, together with an expected result.
- Can be automated.
- Integration tests – Implemented by developers.
- Testing units together within a larger system.
- Can be automated.
- Functional tests – Performed by independent testers.
- Normally following a manual test script.

Design for testing

- Need to think about testing when designing software.
- Software components should be called through function calls.
- Need to separate data and state, such that code can be initialised for a test.
- May need to create mock data to generate test results.
- Need to be able to inject this data into test code.
- Software that is tightly coupled cannot be efficiently tested.
- Cannot define detailed unit tests that mimic normal usage.

Automated testing

- Normally implement automated unit tests.
- Prevent source code from being committed to a software repository when these tests fail.
- Automated build systems can provide test results.
- Allow developers to see what the problem is via a web interface.

Unit tests

MyClass.py : code under test.

```
class MyClass:
    def passThroughFunction(self, variable):
        return variable
```

Unit test program

```
import unittest
import MyClass
from MyClass import MyClass

class Test_MyClass(unittest.TestCase):
    def test_passThroughFunction(self):
        m = MyClass()
        self.assertEqual(1, m.passThroughFunction(1))

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

Output

```
.
----------------------------------
Ran 1 tests in 0.000s

OK
```

Unit tests

Returns 0 on success and 1 on failure.

```
import unittest
import MyClass
from MyClass import MyClass

class Test_MyClass(unittest.TestCase):
    def test_passThroughFunction(self):
        m = MyClass()
        self.assertEqual(1, m.passThroughFunction(1))

    def test_failingFunction(self):
        m = MyClass()
        self.assertTrue(m.failingFunction())

== '__main__':
main()
```

Output

```
F.
======================================================
FAIL: test_failingFunction (__main__.Test_MyClass)
------------------------------------------------------
Traceback (most recent call last):
  File "UnitTest.py", line 14, in test_failingFunction
    self.assertTrue(m.failingFunction())
AssertionError: False is not true

------------------------------------------------------
Ran 2 tests in 0.000s

FAILED (failures=1)
```

Unit tests

- Unit tests should be separate from the code under test.
- Unit tests are not normally distributed with the software.
- Same software repository, but not within distributed package.
- Unit test names should match function or functionality.
- Create a new object or data structure before test.
- Use `assertTrue` or `assertEqual`.
- May call one function and then test each part of the return value or result.

Summary

- Discussed testing.
- Discussed design for testing.
- Introduced unit tests.