**TDD FileLoader v1.0**

In this solution, the production class actually loads a file from the disk.

**Unit Test**

import unittest

from app.file\_loader import FileLoader

class LiveFileLoaderTest(unittest.TestCase):

"""

The initial design is described in this test.

The weakness should be obvious — the file to be loaded and its location.

I use a shared network drive to run this code from different machines, normally

developing from a PC. When I ran the code on the laptop from a cafe it

immediately failed because the C: on the laptop was completely different

to the PC, so the original file C:/tmp/KeyboardHandler.txt did not exist.

THIS IS A GREAT EXAMPLE OF WHY THE UNIT TEST AND CUT SHOULD NOT BE STRONGLY

LINKED TO ANY IO — NETWORK, DB, AND FILE SYSTEM.

"""

def test\_load\_all\_of\_file\_using\_inbuilt\_files\_type(self):

# Arrange

file\_to\_load = "sample.txt"

expected\_bytes\_read = 12

cut = FileLoader(file\_to\_load)

# Act

bytes\_read = cut.load\_file(file\_to\_load)

# Assert

self.assertEqual(expected\_bytes\_read, bytes\_read)

if \_\_name\_\_ == "\_\_main\_\_":

unittest.main()

**The CUT FileLoader**

class FileLoader:

def \_\_init\_\_(self, file\_to\_load):

self.file\_to\_load = file\_to\_load

self.lines = []

def load\_file(self, fname):

try:

with open(fname, encoding='utf-8') as f:

self.lines = f.readlines()

except IOError:

self.lines = []

return self.\_calculate\_file\_size()

def \_calculate\_file\_size(self):

total\_length = sum(len(line) for line in self.lines)

return total\_length