

Maestría en Sistemas Inteligentes Multimedia



NOMBRE DEL DOCENTE

Dr. Héctor Torres Ortega

ALUMNO:

Héctor Edmundo Ramírez Gómez

MATERIA:

Prueba y Validación de Software

ZAPOPAN, JALISCO.

23/03/2018

Introducción

El probar y validar un software es esencial para garantizar su fiabilidad en tiempo de ejecución, pues así se disminuye la probabilidad de que presente fallas inesperadas y costosas para el cliente.

Para este proyecto, se desarrolló uno de los módulos principales del proyecto de tesis del autor. Este módulo se encarga de calcular los volúmenes y precios de algunos productos derivados de madera. El proceso que se siguió fue el siguiente:

- Diseño del algoritmo y codificación
- Diseño y codificación de pruebas
- Automatización de las pruebas
- Generación de reportes

En cada una de estas secciones se mostrará información detallada del proceso, el código fuente y elementos relacionados está disponible en <https://github.com/edmundormz/python-testing>

Diseño y codificación del algoritmo

El proceso iniciación con el desarrollo de un boceto, una representación informal de los requerimientos del software, que se presenta a continuación:

Programa que calcula volúmenes y precios de madera de acuerdo al tipo de producto (aserrío, raja, troncos, celulosas)

Para aserrío recibe:

- ancho (pulgadas)
- alto (pulgadas)
- largo (pies)
- clase (entero [1, 2, 3])

retorna:

- Descripción y precio (string [" 4" x 3/4 " x 8', \$45"])

Para raja recibe:

- ancho carga (metros)
- largo carga (metros)
- 3 altos (metros)

retorna:

-volumen de carga (metros cúbicos [25.272 = 38.88 x 0.65 (factor de conversión)])

- precio (pesos \$[6065.28])

Para troncos recibe

- longitud (metros)

- diámetro o diámetros (dependiendo de la longitud) (metros) (Si longitud > 4, toma promedio de dos diámetros)

- número de piezas (si la longitud es mayor a 4m)

retorna:

- volumen de tronco(s) (metros cúbicos[0.479])

Para celulosas recibe:

- ancho carga (metros)
- largo carga (metros)
- alto carga (metros)
- tipo de producto (1 [astilla], 2[raja])

retorna:

- volumen de carga (metros cúbicos)

- precio (pesos [44.8])

Una vez establecidos los requerimientos, se codificó el software en el lenguaje de programación Python usando el IDE PyCharm. A continuación se muestra el resultado.

```
import math
import sys
def selector(producto, ancho, largo, alto=None, clase=None, piezas=None):
    if producto == "raja":
        raja(ancho, largo, alto)
    elif producto == "aserrio":
        aserrio(ancho, alto, largo, clase)
    elif producto == "troncos":
        troncos(ancho, largo, piezas)
    elif producto == "aserrin" or producto == "astilla":
        celulosas(producto, ancho, largo, alto)
def raja(ancho, largo, altos):
    """
Calcula el volumen de una carga de raja, promediando 3 medidas de la altura y usando un factor de conversion
:param ancho: ancho de la carga, en metros
:param largo: largo de la carga, en metros
:param altos: lista con n altas, en metros
:return: tupla con volumen de carga y precio de la misma
    """
    alto = 0
    for a in altos:
        alto += a
    alto = alto / len(alto)
    volumen = ancho * alto * largo
    vol_str = "{0:.3f}".format(volumen)
    precio = volumen * 0.65 * 320
    precio_str = "{0:.2f}".format(precio)
    print(float(vol_str), float(precio_str))
    return float(vol_str), float(precio_str)
def aserrio(ancho, alto, largo, clase):
    """
Calcula precio de una pieza de madera aserrada (tabla, polin, viga) de acuerdo a su clasificacion de calidad (primera, segunda o tercera)
:param ancho: ancho de la pieza, en pulgadas
:param alto: alto de la pieza, en pulgadas
:param largo: largo de la pieza, en pies
:param clase: calidad de la pieza (1, 2 o 3), determina el precio
:return: string con descripcion y precio de la pieza
    """
    volumen = ancho * largo * alto / 12
    if clase == 1:
        precio = volumen * 15
    if clase == 2:
        precio = volumen * 12
    if clase == 3:
        precio = volumen * 10
    nombre = '{0} x {1} x {2} {3}a, ${4}'.format(str(ancho), str(alto), str(largo), str(clase), str(precio))
```

```

    print nombre
    return nombre
def troncos(diametro, longitud, piezas=1):
    """
    Calcula el volumen de troncos de madera
    :param diametro: diametro de la pieza, en centimetros
    :param longitud: longitud de la pieza, en metros
    :param piezas: cantidad de piezas que comparten diametro y longitud para
    calcular sus volúmenes
    :return: volumen del total de troncos calculados
    """
    radio_metros = (float(diametro) / 100) / 2
    area = math.pi * (radio_metros ** 2)
    volumen = area * longitud * piezas
    vol_str = "{0:.3f}".format(volumen)
    print vol_str
    return float(vol_str)
def celulosas(producto, ancho, largo, alto):
    """
    Calcula volumen y costo de cargas de productos para celulosa,
    como aserrin o astilla
    :param producto: tipo de producto [aserrin, astilla] para definir precio
    :param ancho: ancho de la carga, en metros
    :param largo: largo de la carga, en metros
    :param alto: longitud de la carga, en metros
    :return: tupla que contiene volumen calculado y precio del mismo
    """
    volumen = ancho * largo * alto
    if producto == "astilla":
        precio = volumen * 240
    elif producto == "aserrin":
        precio = volumen * 180
    else:
        msj = "Producto incorrecto"
        print msj
        return msj
    vol_str = "{0:.3f}".format(volumen)
    print(vol_str, precio)
    return float(vol_str), precio
if __name__ == "__main__":
    selector(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4], sys.argv[5])

```

Durante el desarrollo de este software se respetaron las reglas del estándar PEP8, que es el equivalente de Misra C++. Aquí se muestra el resultado de uno de los análisis:

```
Terminal
+ hecmundo@MenLoPark:~/PycharmProjects/python-testing$ pep8 --statistics madera.py
X madera.py:5:36: E231 missing whitespace after ','
madera.py:32:30: E225 missing whitespace around operator
madera.py:56:60: E231 missing whitespace after ','
madera.py:72:45: E225 missing whitespace around operator
madera.py:80:30: E231 missing whitespace after ','
2      E225 missing whitespace around operator
3      E231 missing whitespace after ','
hecmundo@MenLoPark:~/PycharmProjects/python-testing$
```

Nótese cómo la última línea indica que hay tres incidencias que violan la regla E231 “missing whitespace after ‘,’ character.

Diseño y codificación de pruebas

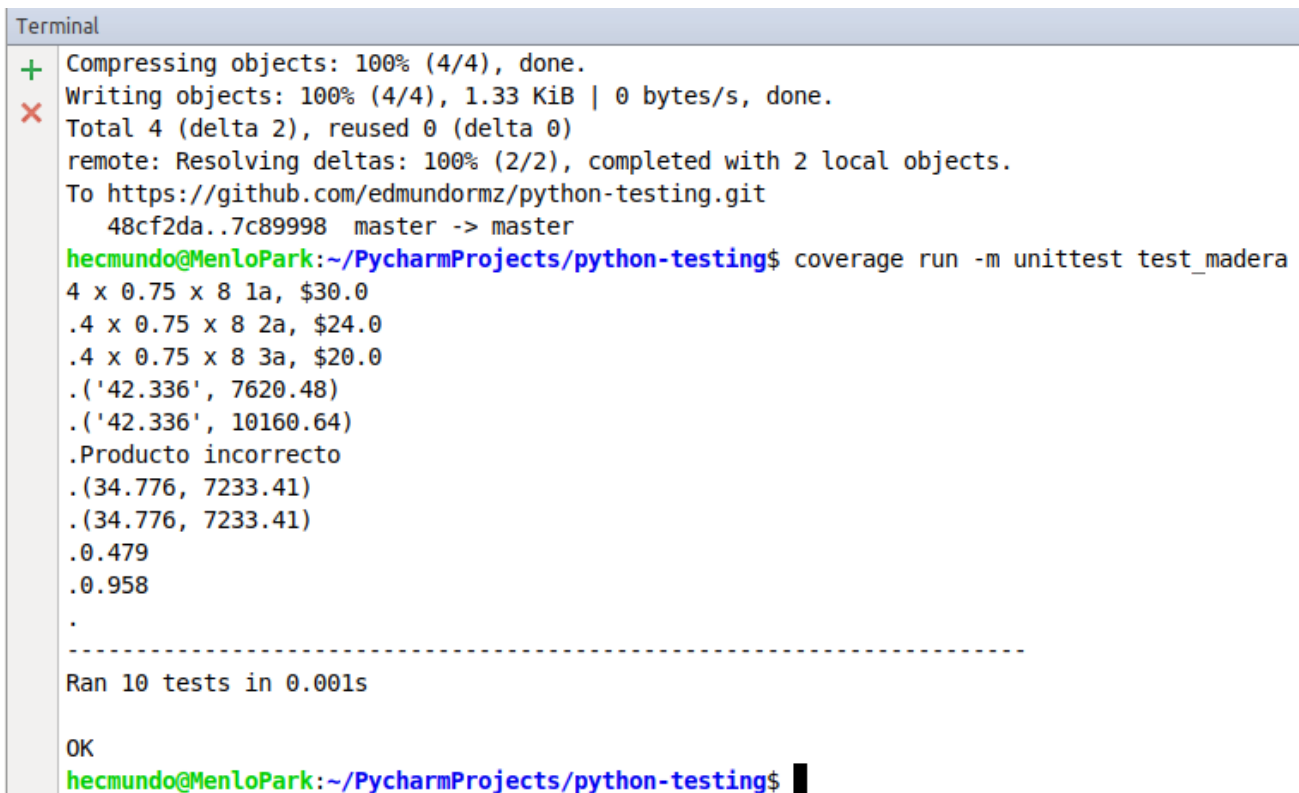
Las pruebas se diseñaron bajo la metodología de cobertura, automatizándolas con herramientas como coverage y unittesting. A continuación se muestra el código de pruebas.

```
import unittest
import madera
class TestCelulosas(unittest.TestCase):
    def test_celulosas_aserrin(self):
        self.assertEqual(madera.celulosas("aserrin", 2.4, 6.3, 2.8),
                         (42.336, 7620.48))
    def test_celulosas_astilla(self):
        self.assertEqual(madera.celulosas("astilla", 2.4, 6.3, 2.8),
                         (42.336, 10160.64))
    def test_celulosas_invalido(self):
        self.assertEqual(madera.celulosas("bombones", 0, 0, 0),
                         "Producto incorrecto")
class TestTroncos(unittest.TestCase):
    def test_troncos_no_pieces(self):
        self.assertEqual(madera.troncos(50, 2.44), 0.479)
    def test_troncos_two_pieces(self):
        self.assertEqual(madera.troncos(50, 2.44, 2), 0.958)
class TestAserrio(unittest.TestCase):
    def test_aserrio_1a(self):
        self.assertEqual(madera.aserrio(4, 0.75, 8, 1),
                         "4 x 0.75 x 8 1a, $30.0")
    def test_aserrio_2a(self):
        self.assertEqual(madera.aserrio(4, 0.75, 8, 2),
                         "4 x 0.75 x 8 2a, $24.0")
    def test_aserrio_3a(self):
        self.assertEqual(madera.aserrio(4, 0.75, 8, 3),
                         "4 x 0.75 x 8 3a, $20.0")
```

```
class TestRaja(unittest.TestCase):
    def test_raja_1_alto(self):
        self.assertEqual(madera.raja(2.4, 6.3, [2.3]),
                          (34.776, 7233.41))
    def test_raja_3_altos(self):
        self.assertEqual(madera.raja(2.4, 6.3, [2.3, 2.6, 2]),
                          (34.776, 7233.41))
```

Automatización de pruebas

El código anterior, que define las pruebas, se automatizó para correr y generar un reporte de cobertura en html. La siguiente imagen muestra cómo todas las pruebas fueron exitosas.



Terminal

```
+ Compressing objects: 100% (4/4), done.
x Writing objects: 100% (4/4), 1.33 KiB | 0 bytes/s, done.
Total 4 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/edmundormz/python-testing.git
   48cf2da..7c89998  master -> master
hecmundo@MenLoPark:~/PycharmProjects/python-testing$ coverage run -m unittest test_madera
4 x 0.75 x 8 1a, $30.0
.4 x 0.75 x 8 2a, $24.0
.4 x 0.75 x 8 3a, $20.0
.('42.336', 7620.48)
.('42.336', 10160.64)
.Producto incorrecto
.(34.776, 7233.41)
.(34.776, 7233.41)
.0.479
.0.958
.
-----
Ran 10 tests in 0.001s

OK
hecmundo@MenLoPark:~/PycharmProjects/python-testing$
```

Generación de reportes

La ejecución de estas pruebas se traduce en un porcentaje de cobertura del código. Nótese que el script `test_madera.py` tiene un 100% de cobertura debido a que es el código que contiene las pruebas, en cambio el script `madera.py`, que es el código principal, tiene una cobertura del 89% usando las pruebas diseñadas.

Coverage report: 89%				
<input type="text" value="filter..."/>				
Module ↓	statements	missing	excluded	coverage
madera.py	54	9	0	83%
test_madera.py	26	0	0	100%
Total	80	9	0	89%
coverage.py v4.5.1, created at 2018-03-23 15:09				

Y a continuación se muestra información detallada de las líneas de código que son cubiertas por las pruebas.

Coverage for madera.py : 83%				
54 statements	45 run	9 missing	0 excluded	
1	<code>import math</code>			
2	<code>import sys</code>			
3				
4				
5	<code>def selector(producto, ancho, largo, alto=None, clase=None, piezas=None):</code>			
6	<code> if producto == "raja":</code>			
7	<code> raja(ancho, largo, alto)</code>			
8	<code> elif producto == "aserrio":</code>			
9	<code> aserrio(ancho, alto, largo, clase)</code>			
10	<code> elif producto == "troncos":</code>			
11	<code> troncos(ancho, largo, piezas)</code>			
12	<code> elif producto == "aserrin" or producto == "astilla":</code>			
13	<code> celulosas(producto, ancho, largo, alto)</code>			
14				
15				
16	<code>def raja(ancho, largo, altos):</code>			
17	<code> """</code>			
18	<code> Calcula el volumen de una carga de raja, promediando 3 medidas de la</code>			
19	<code> altura y usando un factor de conversion</code>			
20				
21	<code> :param ancho: ancho de la carga, en metros</code>			
22	<code> :param largo: largo de la carga, en metros</code>			
23	<code> :param altos: lista con n alturas, en metros</code>			
24	<code> :return: tupla con volumen de carga y precio de la misma</code>			
25	<code> """</code>			
26	<code> alto = 0</code>			
27	<code> for a in altos:</code>			
28	<code> alto += a</code>			
29	<code> alto = alto / len(alto)</code>			
30	<code> volumen = ancho * alto * largo</code>			
31	<code> vol_str = "{0:.3f}".format(volumen)</code>			
32	<code> precio = volumen * 0.65 * 320</code>			
33	<code> precio_str = "{0:.2f}".format(precio)</code>			
34	<code> print(float(vol_str), float(precio_str))</code>			
35	<code> return float(vol_str), float(precio_str)</code>			
--				


```

38 | def aserrio(ancho, alto, largo, clase):
39 |     """
40 |     Calcula precio de una pieza de madera aserrada (tabla, polin, viga) de
41 |     acuerdo a su clasificacion de calidad (primera, segunda o tercera)
42 |     :param ancho: ancho de la pieza, en pulgadas
43 |     :param alto: alto de la pieza, en pulgadas
44 |     :param largo: largo de la pieza, en pies
45 |     :param clase: calidad de la pieza (1, 2 o 3), determina el precio
46 |     :return: string con descripcion y precio de la pieza
47 |     """
48 |     volumen = ancho * largo * alto / 12
49 |     if clase == 1:
50 |         precio = volumen * 15
51 |     if clase == 2:
52 |         precio = volumen * 12
53 |     if clase == 3:
54 |         precio = volumen * 10
55 |     nombre = '{0} x {1} x {2} {3}a, ${4}'.format(str(ancho), str(alto),
56 |                                                  str(largo), str(clase),
57 |                                                  str(precio))
58 |     print nombre
59 |     return nombre
60 |
61 |
62 | def troncos(diametro, longitud, piezas=1):
63 |     """
64 |     Calcula el volumen de troncos de madera
65 |     :param diametro: diametro de la pieza, en centimetros
66 |     :param longitud: longitud de la pieza, en metros
67 |     :param piezas: cantidad de piezas que comparten diametro y longitud para
68 |     calcular sus volúmenes
69 |     :return: volumen del total de troncos calculados
70 |     """
71 |     radio_metros = (float(diametro) / 100) / 2
72 |     area = math.pi * (radio_metros ** 2)
73 |     volumen = area * longitud * piezas
74 |     vol_str = "{0:.3f}".format(volumen)
75 |     print vol_str
76 |     return float(vol_str)
77 |
78 |
79 | def celulosas(producto, ancho, largo, alto):
80 |     """
81 |     Calcula volumen y costo de cargas de productos para celulosa,
82 |     como aserrin o astilla
83 |     :param producto: tipo de producto [aserrin, astilla] para definir precio
84 |     :param ancho: ancho de la carga, en metros
85 |     :param largo: largo de la carga, en metros
86 |     :param alto: longitud de la carga, en metros
87 |     :return: tupla que contiene volumen calculado y precio del mismo
88 |     """
89 |     volumen = ancho * largo * alto
90 |     if producto == "astilla":
91 |         precio = volumen * 240
92 |     elif producto == "aserrin":
93 |         precio = volumen * 180
94 |     else:
95 |         msj = "Producto incorrecto"
96 |         print msj
97 |         return msj
98 |     vol_str = "{0:.3f}".format(volumen)
99 |     print(vol_str, precio)
100 |    return float(vol_str), precio
101 |
102 |
103 | if __name__ == "__main__":
104 |     selector(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4], sys.argv[5])

```

Complejidad ciclomatica

Finalmente, la complejidad ciclomática promedio de todo el programa fue calculada, tomando en cuenta las complejidades parciales de cada función

```
Terminal
+ 1      E902 IOError: [Errno 2] No such file or directory: '8'
x hecmundo@MenloPark:~/PycharmProjects/python-testing$ radon cc madera.py -a
madera.py
  F 5:0 selector - B
  F 38:0 aserrio - A
  F 79:0 celulosas - A
  F 16:0 raja - A
  F 62:0 troncos - A

5 blocks (classes, functions, methods) analyzed.
Average complexity: A (3.2)
hecmundo@MenloPark:~/PycharmProjects/python-testing$
```