Министерство образования Российской Федерации

Тверской государственный технический университет

Кафедра Программного обеспечения

Программа для работы с личными делами студентов

Unit-тестирование

Выполнил: Бакеев А.В.

Проверил: Мальков А.А

Тверь, 2024

Оглавление

[Введение 3](#_Toc181692025)

[Постановка задачи 3](#_Toc181692026)

[Классы для unit-тестирования 4](#_Toc181692027)

[Таблица тестов 4](#_Toc181692028)

[Программа тестирования стороны студента 5](#_Toc181692029)

[Программа тестирования стороны университета 6](#_Toc181692030)

# Введение

Даже если вы никогда в жизни не думали, что занимаетесь тестированием, вы это делаете. Вы собираете свое приложение, нажимаете кнопку и проверяете, соответствует ли полученный результат вашим ожиданиям. Достаточно часто в приложении можно встретить формочки с кнопкой “Test it” или классы с названием *TestController* или *MyServiceTestClient*.

То что вы делаете, называется *интеграционным тестированием*. Современные приложения достаточно сложны и содержат множество зависимостей. Интеграционное тестирование проверяет, что несколько компонентов системы работают вместе правильно.

# Постановка задачи

**Вариант 2.** Разработать программный модуль «Личные дела студентов». Программный модуль предназначен для получения сведений о студентах сотрудниками деканата, профкома и отдела кадров. Сведения должны храниться в течение всего срока обучения студентов и использоваться при составлении справок и отчетов.

Провести unit-тестирование разработанного ПО, согласно варианту задания.

Отчет о проделанной работе предоставить преподавателю.

Отчет должен содержать:

- Разработанные и написанные классы согласно варианту задания

- Таблицу с набором тестов

- Программу, выполняющую тестовые методы и формирующую выходные данные.

# Классы для unit-тестирования

1. TestStudentApp(unittest.TestCase) – класс тестирования той части, которую видит студент. В скобочках указан родительский класс, от которого наследуется (синтаксис пайтона). Является примером тестирования белого ящика, т.к мы не трогаем в нем пользовательскую часть вообще, а работаем исключительно с кодом приложения.
2. TestUniversityApp(unittest.TestCase) - класс тестирования той части, которую видит сотрудник. В скобочках указан родительский класс, от которого наследуется (синтаксис пайтона). Тестирование серого ящика, т.к. внутри реализованы методы тестирования как для кода, так и для пользовательской части приложения.

# Таблица тестов

|  |  |  |  |
| --- | --- | --- | --- |
| Требование | Тест | Ожидание | Комментарии |
| Корректное закрытие приложение | Вызвать метод doSave после нажатия кнопки | Закрытие приложения (Выход из него) | Проверяем, что приложение завершает все окна без ошибок |
| Количество кнопок, сгенерировавшихся на форме | Проверить соответствие количества кнопок | Одна кнопка генерируется и отображается на форме | Проверяем, что генерация кнопки осуществляется как нам нужно |
| Количество полей, сгенерировавшихся на форме | Проверить соответствие количества полей | 4 поля (по 1 для каждого реквизита: ФИО, город рождения, средний балл, статус о работе) | Проверяем, что для каждого реквизита существует поле |
| Корректное открытие формы записи данных | Вызывать метод goWrite после нажатия кнопки | Основная форма открыта и доступна для просмотра. Поверх нее открывается форма получения данныых | Проверяем, что дополнительная форма получения данных (и генерации справок и отчетов) открывается в нужном варианте |

# Программа тестирования стороны студента

import unittest  
from unittest.mock import patch  
from student import WriteInfo  
from PyQt5.QtWidgets import QApplication  
  
  
class TestStudentApp(unittest.TestCase):  
  
 def setUp(self):  
 self.app = QApplication([]) # Создаем экземпляр QApplication  
 self.student\_app = WriteInfo() # Создаем экземпляр нашего приложения  
  
 @patch.object(WriteInfo, 'close')  
 def test\_do\_save\_closes\_window(self, mock\_close):  
 self.student\_app.doSave() # Вызываем метод doSave  
 mock\_close.assert\_called\_once() # Проверяем, что close был вызван один раз (соответственно отработало)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

# Программа тестирования стороны университета

import sys  
import unittest  
from PyQt5.QtWidgets import QApplication, QPushButton, QLabel  
from student import ReadInfo, WriteInfo # Импортируйте классы из вашего основного файла  
  
class TestUniversityApp(unittest.TestCase):  
  
 @classmethod  
 def setUpClass(cls):  
 # Этот метод запустится один раз перед всеми тестами  
 cls.app = QApplication(sys.argv) # Создаём экземпляр QApplication  
 cls.form = ReadInfo() # Создаем экземпляр формы  
  
 def test\_button\_count(self):  
 # Проверяем количество кнопок на форме  
 buttons = [child for child in self.form.children() if isinstance(child, QPushButton)]  
 self.assertEqual(len(buttons), 1, "Количество кнопок на форме должно быть 2")  
  
 def test\_field\_count(self):  
 # Проверяем количество полей на форме  
 fields = [child for child in self.form.children() if isinstance(child, QLabel)]  
 self.assertEqual(len(fields), 4, "Количество кнопок на форме должно быть 2")  
  
 def test\_open\_write\_info(self):  
 # Проверяем, что открытие формы WriteInfo работает корректно  
 self.form.goWrite() # Открываем форму WriteInfo  
 write\_info\_widget = self.form.studentWrite # Получаем ссылку на созданный виджет  
 self.assertIsInstance(write\_info\_widget, WriteInfo, "Форма WriteInfo не была создана")  
  
 @classmethod  
 def tearDownClass(cls):  
 # Этот метод запустится один раз после всех тестов  
 cls.form.close() # Закрываем форму  
 del cls.form # Удаляем форму  
 del cls.app # Удаляем экземпляр приложения  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()