

## 5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование программного обеспечения – это процесс исследования программного обеспечения, который преследует 2 цели:

- проверить программный продукт на соответствие заявленным требованиям;
- выявить условия, в которых поведение программы является ошибочным либо не соответствует спецификации;

Тестирование программного обеспечения обычно проводится в несколько этапов и является таким же важным этапом разработки программного обеспечения, как и написание кода. Хорошо отлаженный процесс тестирования позволяет как обнаружить уже существующие ошибки, так и не допустить появления новых в уже отлаженном рабочем коде.

Помимо рисков, связанных с неизбежным появлением ошибок в программном коде, существует также риск написать программу, не соответствующую заявленным к ней требованиям. С этой стороны, целью тестирования является выявление несоответствия функционирования программы заявленным требованиям, и устранение этих несоответствий с целью повышения качества.

Тестирование проводилось в 3 этапа:

- модульное тестирование, которое проводилось в процессе написания программного кода;
- автоматическое интеграционное тестирование, которое проводилось после того как все модули программы были в рабочем состоянии;
- полное функциональное тестирование программы после окончания написания программного кода;

Все три этапа важны для обеспечения качества кода. Модульное тестирование позволяет описать желаемое поведение небольших частей кода, функций и методов, и даёт программисту уверенность в том, что ошибок в протестированной части кода нет. Также модульное тестирование позволяет быстро удостовериться в том, что после внесения изменений в код программы он всё ещё работает корректно. Недостаток модульного тестирования в том, что он не гарантирует корректную работу всех модулей вместе. Для того что бы дать программисту такую гарантию, нужны автоматические интеграционные тесты, выполнение которых занимает чуть больше времени и требует работоспособности всех модулей программы. Полное функциональное тестирование программного обеспечения позволяет выявить те недостатки, которые нельзя выявить на предыдущих этапах тестирования, а именно:

- неудобность использования программы для пользователя;
- медленная работа программы в рамках наиболее частых сценариев

взаимодействия программы и пользователя;

- побочные эффекты после выполнения программой той или иной команды пользователя;

Тестирование программы проводилось на следующих машинах:

- 1) AMD Phenom II X4 965 4 ядра по 3,4 ГГц, оперативная память 8Гб, видеокарта nVidia GTX260 на 1Гб. Диск HDD на 1Тб, скорость - 7200 об/мин. Операционная система - Manjaro Linux.
- 2) Intel Core i5-4200 4 ядра по 2,4 ГГц, оперативная память 8 Гб, видеокарта nVidia GeForce 960M. Диск SSD на 500 Гб. Операционная система - Ubuntu Linux.
- 3) Intel Core i7-3770 8 ядер по 3,4 ГГц, оперативная память 16 Гб, видеокарта nVidia GeForce 960M. Диск HDD на 1Тб, скорость – 5200 об/мин. Операционная система – OpenSUSE Linux.

## 5.1 Модульное тестирование

Модульное тестирование помогает проверить корректность программы путём проверки поведения небольших участков кода, таких как методы и функции. Тест одного из аспектов поведения модуля обычно состоит из трёх этапов:

- моделирование окружения;
- произведение запланированного действия;
- проверка результатов этого действия;

Наиболее сложным этапом создания модульного теста является моделирование окружения. Поскольку каждый тест должен быть изолирован от остальных, окружение настраивается также индивидуально. При настройке задаются все начальные условия, подменяются возвращаемые значения для вызываемых модулем функций, определяются параметры для вызываемого модуля, если они имеют сложную структуру и т.д. Далее над модулем производится действие, например, вызов метода с заранее заданными параметрами. И в завершении проверяется результат вызова – это может быть, как проверка возвращаемого значения метода или функции, так и проверка состояния окружения, если работа метода или функции имеет запланированный побочный эффект.

Написав серию тестов, по одному на каждый аспект работы модуля, можно сказать что модуль покрыт тестами. При внесении в него изменений можно проверить, нарушили ли новые изменения уже отлаженную логику или нет.

Каждый из популярных языков программирования имеет свою библиотеку для проведения модульного тестирования. В языке Python за это отвечает модуль unittest, который и используется для тестирования данного приложения. Пример

класса, содержащего тесты для методов класса `CommandsDescriptionModule`:

```
class CommandsDescriptionModuleTest(unittest.TestCase):
    def setUp(self):
        self.commands_module = CommandsDescriptionModule()

    def test_executing_none_command(self):
        with self.assertRaises(ValueError):
            self.commands_module.execute_command(None)

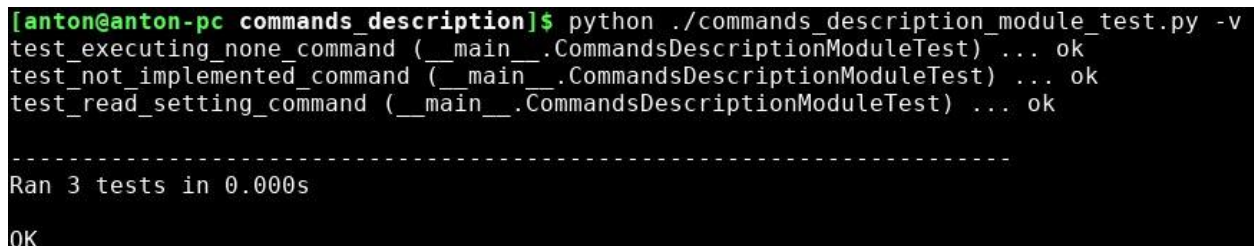
    def test_not_implemented_command(self):
        with self.assertRaises(NotImplementedError):
            bad_command = Command(tbt.EVENT_LOG_BLOCK, \
                                   att.CREATE, None)
            self.commands_module.execute_command(\
                bad_command)

    def test_read_setting_command(self):
        read_command = Command(tbt.SETTINGS_BLOCK, \
                                att.READ, None)
        setting = self.commands_module.execute_command(\
            read_command)
        key, value = setting

        self.assertIs(type(setting), tuple)
        self.assertIsNotNone(key)
        self.assertIsNotNone(value)
```

В методе `setUp` задаются общие начальные условия для всех тестов, которые содержатся в данном классе как методы. Этот метод выполняется перед выполнением каждого теста. Методами формата `assert*` проверяются результаты выполнения действий.

Если запустить модуль, содержащий данный класс тестов, в консоль выводится результат выполнения всех тестов в классах этого модуля. В случае класса `CommandsDescriptionModuleTest` вывод представлен на рисунке 5.1. Тесты также можно запускать и по отдельности, но делается это обычно если их настолько много, что их полное выполнение занимает длительное время.



```
[anton@anton-pc commands_description]$ python ./commands_description_module_test.py -v
test_executing_none_command (__main__.CommandsDescriptionModuleTest) ... ok
test_not_implemented_command (__main__.CommandsDescriptionModuleTest) ... ok
test_read_setting_command (__main__.CommandsDescriptionModuleTest) ... ok
-----
Ran 3 tests in 0.000s
OK
```

Рисунок 5.1 – Вывод результата запуска модульных тестов

## 5.2 Интеграционное тестирование

Интеграционное тестирование – это фаза тестирования, при которой модули объединяются и тестируются в группе. Обычно интеграционное тестирование следует за модульным и предшествует полному системному тестированию. Интеграционное тестирование позволяет выявить ошибки взаимодействия модулей, задать набор приёмочных тестов, без прохождения которых невозможен выпуск приложения и протестировать приложение под нагрузкой.

Поскольку данное приложение является тремя взаимодействующими между собой программами, цель которых в автоматическом режиме распределять файлы по заданным пользователем правилам – интеграционное тестирование имеет для него первостепенную значимость. Интеграционные тесты для данного приложения написаны на языке оболочки `bash`. Интеграционный тест, так же, как и модульный состоит из задания начальных условий, действия и проверки результата. Примеры интеграционных тестов для проверки правил распределения файлов выглядят следующим образом:

```
# Test 24. Permanently delete file if name changed

mkdir ./test
touch ./test/file.txt
tars add_rule -t /home/anton/Tars/Program/test/ \
    --event-types FILE_NAME_CHANGED \
    --extention-template txt \
    --action-type DELETE_FILE --is-permanent
mv ./test/file.txt ./test/file2.txt

if [ -f ./test/file2.txt ]
then
    echo "ERROR. File ./test/file2.txt was not deleted."
else
    echo "SUCCESS. File ./test/file2.txt was deleted."
fi

rm -rf ./test

# Test 25. Peplace file in subfolder if file moved from

mkdir -p ./test/sub
touch file.txt
tars add_rule -t /home/anton/Tars/Program/test/ \
    --event-types FILE_INCLUDED \
    --extention-template txt \
```

```

        --action-type REPLACE_FILE \
        --target-path /home/anton/Tars/Program/test/sub/
mv file.txt ./test/

if ! [ -f ./test/sub/file.txt ]
then
    echo "ERROR. File file.txt was not found in
./test/sub/."
else
    echo "SUCCESS. File file.txt was not found in
./test/sub/."
fi

rm -rf ./test/

```

В этих двух тестах вначале создаётся директория, действия в которой будут отслеживаться приложением, создаётся тестовый файл, задаётся правило распределения и над тестовым файлом производится действие таким образом, чтобы он попал под заданное правило. После того как приложение среагировало на событие в файловой системе – в bash-скрипте идёт проверка того что стало с файлом. В зависимости от результатов проверки в консоль выводится либо сообщение об успешно выполненной операции, либо с сообщением об ошибке. Таким образом, имея доступ управлению программой из консоли, можно эмулировать практически любые действия пользователя.

### 5.3 Полное функциональное тестирование программы

Полное функциональное тестирование приложения проводится после того как код программы полностью написан, все модульные и интеграционные тесты пройдены и нужно решить можно ли продавать/передавать приложение его заказчику или конечному пользователю. Полное тестирование программы проводится людьми, чтобы понять соответствует ли программа заявленным требованиям, какие недостатки имеет интерфейс пользователя, не проявляется ли просадка производительности при типичных сценариях взаимодействия и не оставляет ли работа приложения неучтённых побочных эффектов.

Перед полным тестированием программы составляется спецификация, состоящая из требований, которые оформляются в виде тестовых случаев (Test Cases) и бизнес-процессов, которые описываются в виде типичных сценариев взаимодействия пользователя и программы (Use Cases).

Тестирование проводилось несколькими пользователями на разных компьютерах с разной аппаратной конфигурацией и разными операционными системами. Функциональные тесты, проведённые над программным модулем

представлены в таблице 5.1.

Таблица 5.1 – Полное функциональное тестирование программы

Модуль	Содержание теста	Ожидаемый результат	Тест пройден
1	2	3	4
Настройки	Создание настройки без задания значения	Сообщение о некорректной команде	Да
Настройки	Обновление настройки без задания значения	Сообщение о некорректной команде	Да
Настройки	Удаление обязательной настройки приложения	Сообщение о невозможности удалить настройку	Да
Настройки	Экспорт настроек в существующий системный файл	Сообщение о невозможности экспортировать настройки в данный файл	Да
Настройки	Импорт настроек из пустого файла	Сообщение с перечислением обязательных настроек	Да
Распределение файлов	Перемещение в отслеживаемую директорию большого количества файлов разных типов	Файлы распределяются в течении небольшого промежутка времени	Да
Распределение файлов	Перемещение в отслеживаемую директорию файлов, которые должны игнорироваться	С перемещёнными файлами ничего не происходит	Да
Распределение файлов	Создание правила распределения без обязательных аргументов	Сообщение с перечислением пропущенных аргументов	Да

Продолжение таблицы 5.1

1	2	3	4
Распределение файлов	Удаление и создание отслеживаемой директории	Правила для директории удаляются после удаления директории	Да
Распределение файлов	Импорт правил из пустого файла	Сообщение об ошибке импорта с указанием того что файл пуст	Да
Анализ файлов	Изменения расширения файла на расширение файла с другим типом контента	Расширение не влияет на распознавание типа контента	Да
Анализ файлов	Изменение метаданных файла	Файл распределяется по новым метаданным	Да
Анализ файлов	Изменение размера файла	Файл распределяется по новому размеру	Да
Анализ файлов	Задание правила для удаления системного файла	Файл распознаётся как системный, в файл журнала записывается ошибка о невозможности применить правило к системному файлу	Да
Анализ файлов	Создание директории, которая подпадает под одно из правил только для регулярных файлов	Директория распознаётся, правило к ней не применяется	Да

Как видно из таблицы, приложение прошло все функциональные тесты, что говорит о его готовности к выпуску.