Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение   
высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

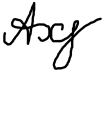
Кафедра компьютерных систем в управлении и проектировании (КСУП)

**ЮНИТ-ТЕСТИРОВАНИЕ**

Отчет по лабораторной работе   
по дисциплине «Новые технологии в программировании»

Вариант №1

Студент гр. 588-1

 \_\_\_\_\_\_\_\_\_ Д.С. Ахроменко

«17» 06 2021

Руководитель

доцент кафедры КСУП

\_\_\_\_\_\_\_\_\_ А.Е. Горяинов  
 « » 2021

Томск 2021

# ЛАБОРАТОРНАЯ РАБОТА №4

Юнит-тестирование

# Цель работы

Целью данной лабораторной работы является изучение организации тестирования в разработке ПО и получение умения написания юнит-тестов.

# Постановка задачи

1. Изучить организацию процесса тестирования в разработке ПО, виды тестирования, сроки их проведения и ответственных исполнителей.
2. Изучить основные атрибуты и классы библиотеки NUnit для написания юнит-тестов.
3. Научиться рассчитывать цикломатическую сложность методов, классов и проектов, оценивать степень покрытия кода тестами.
4. Написать юнит-тесты для классов логики приложения с использованием библиотеки NUnit.

# Дерево цикломатической сложности методов и классов проекта бизнес-логики приложения

***Юнит-тестирование*** – это тестирование программы в виде отдельных, изолированных друг от друга минимальных модулей. Условие изоляции тестируемого модуля от других необходимо для того, чтобы в случае обнаружения ошибки быть уверенным, что ошибка возникла именно в тестируемом модуле. В противном случае найти место возникновении ошибки будет гораздо сложнее.

В процедурном программировании минимальными модулями для юнит-тестирования являются функции или процедуры. В ООП минимальными модулями считаются классы, а не методы, так как поведение методов может зависеть от текущего состояния экземпляра класса. Таким образом, юнит-тесты в ООП – это тесты, выполняющие тестирование отдельных классов с максимально возможной изоляцией от поведения других классов.

Юнит-тестированию подлежит поведение класса, находящееся под модификаторами доступа public или protected. Закрытая реализация класса (private) напрямую тестированию не подвергается, только опосредованно через вызов открытых методов класса. Это обусловлено тем, что юнит-тесты фактически имитируют случаи реального использования классов логики, а, следовательно, не должны нарушать инкапсуляцию тестируемого класса.

***Цикломатической сложностью*** называют количество линейно независимых алгоритмических маршрутов через программный код. Расчет цикломатической сложности позволяет определить все требуемые тесты (тестовые случаи) и исходные данные, позволяющие полностью покрыть исходный код тестами.

При определении цикломатической сложности метода учитывается все – составные условия, наличие циклов, операторов break и continue, вызов других методов. Если внутри тестируемого метода вызывается другой метод, то это также необходимо учитывать в значении цикломатической сложности.

Можно понять, что цикломатическая сложность будет расти при большей иерархии вызова методов внутри класса, а также при вызове методов агрегируемых объектов. Чем выше класс стоит в иерархии агрегирования, тем большую цикломатическую сложность (теоретически) имеют его методы. Это отражается в необходимости большого числа тестов для качественного и полного тестирования кода.

Для приложения NoteApp необходимо выполнить тестирование трех классов бизнес-логики – Note, Project и ProjectManager. Их цикломатическая сложность определяется как сумма сложностей каждого метода соответствующего класса. Далее, исходя из цикломатической сложности данных классов, рассчитывается цикломатическая сложность всего проекта, которая может быть представлена в виде дерева следующим образом:

Проект NoteApp (цикл. сложность проекта = 45)

1. Класс Note (цикл. сложность класса = 28)

Свойство Title (цикл. сложность = 4)

Свойство Category (цикл. сложность = 2)

Свойство Text (цикл. сложность = 2)

Свойство CreationTime (цикл. сложность = 2)

Свойство ModificationTime (цикл. сложность = 2)

Конструктор Note() (цикл. сложность = 1)

Конструктор Note(NoteCategory (цикл. сложность = 1)

category)

Конструктор Note(string title, (цикл. сложность = 7)

NoteCategory category, string text,

DateTime creationTime, DateTime

modificationTime)

Метод Clone() (цикл. сложность = 7)

2. Класс Project (цикл. сложность класса = 9)

Свойство Notes (цикл. сложность = 2)

Свойство CurrentNoteIndex (цикл. сложность = 2)

Метод SortByModificationTime() (цикл. сложность = 2)

Метод SortByModificationTimeAndCategory() (цикл. сложность = 3)

3. Класс ProjectManager (цикл. сложность класса = 8)

Свойство DefaultPath (цикл. сложность = 2)

Метод SaveToFile(Project) (цикл. сложность = 2)

Метод LoadFromFile() (цикл. сложность = 4)

Общее количество юнит-тестов, написанных для данного проекта, – 26. Покрытие кода тестами – 100%.

# Исходный код одного из юнит-тестов

В процессе проверки и отладки программы бывает необходимо знать время выполнения автоматических проверок. Пакеты библиотеки NUnit позволяют наблюдать не только статус тестов, но и время, затраченное на их выполнение.

На рисунке 1 представлена информация о результатах запуска юнит-тестов для проекта бизнес-логики приложения NoteApp.

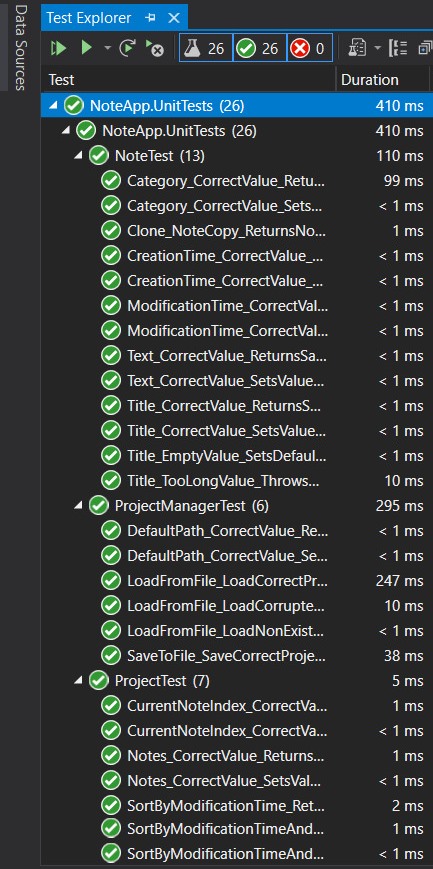


Рисунок 1 – Результат выполнения юнит-тестов

В качестве примера юнит-теста рассмотрим негативный тест Title\_TooLongValue\_ThrowsArgumentException(), проверяющий генерацию исключения типа ArgumentException в сеттере свойства Title при присвоении неправильного названия заметки, превышающего 50 символов (рисунок 2).

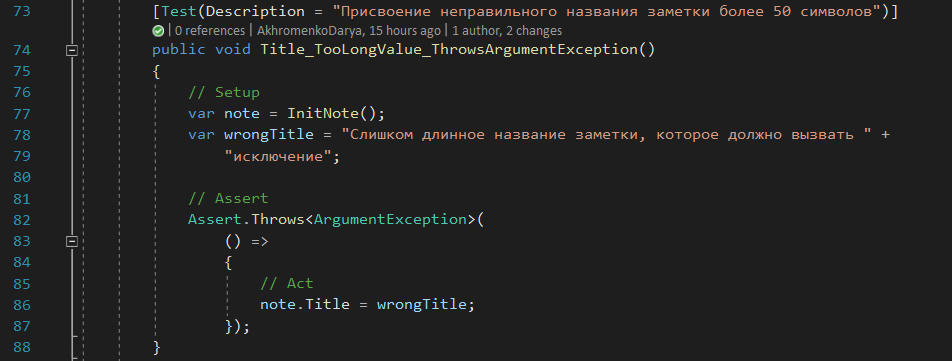


Рисунок 2 – Скриншот исходного кода негативного юнит-теста для сеттера свойства Title

# Текущая история коммитов ветки develop

По итогу выполнения данной лабораторной работы текущая история изменений ветки develop в удаленном репозитории на сервисе GitHub выглядит следующим образом (рисунок 3).

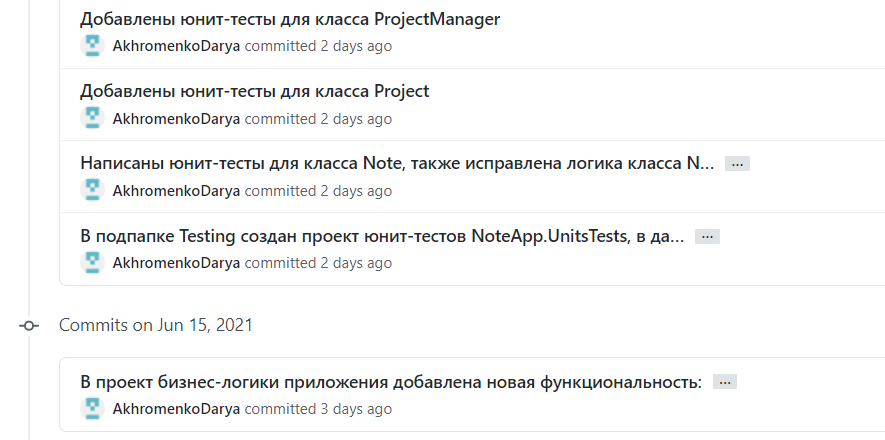


Рисунок 3 – Текущая история коммитов ветки develop

# Вывод

В данной лабораторной работе изучена организация тестирования в разработке ПО и получены умения написания юнит-тестов.