АРХАНГЕЛЬСКИЙ КОЛЛЕДЖ ТЕЛЕКОММУНИКАЦИЙ
ИМ. Б.Л. РОЗИНГА (ФИЛИАЛ) СПбГУТ
(АКТ (ф) СПбГУТ)

Отчеты по лабораторным и практическим работам ПиТПМ

Студент ИСПП-35 *Туйкова А.Е*
(Группа) (Подпись) (Дата) (И.О. Фамилия)
Преподаватель *Садовский Р. В.*
(Подпись) (Дата) (И.О. Фамилия)

Архангельск 2024

Лабораторная работа №6
Модульное тестирование

1 Цель работы

1.1 Тестирование программного кода по методологии белого ящика;

Репозиторий: https://github.com/kkerra/UnitTesting

2 Контрольные вопросы

2.1 Для чего используется атрибут [Fact]?

Атрибут [Fact] в xUnit используется для обозначения тестового метода, который не принимает параметров и предназначен для выполнения единственного тестового случая. Этот атрибут применяется к методам, которые проверяют конкретное поведение или результат без использования входных данных из вне.

2.2 Для чего используется атрибут [Theory]?

Атрибут [Theory] в xUnit используется для обозначения параметризованного тестового метода, который может принимать различные наборы входных данных. Метод, помеченный [Theory], обычно сопровождается атрибутами [InlineData], [MemberData] или другими атрибутами, предоставляющими данные для тестирования. Это позволяет выполнять один тест с разными наборами входных данных.

2.3 Какие методы используются в Assert и для чего они предназначены?

Assert - это статический класс в xUnit, предоставляющий множество методов для проверки условий в тестах. Вот некоторые из наиболее часто используемых методов:

- Assert.Equal(expected, actual): Проверяет, что два значения равны. Если они не равны, тест не проходит.
- Assert.NotEqual(expected, actual): Проверяет, что два значения не равны.
- Assert.True(condition): Проверяет, что условие истинно.
- Assert.False(condition): Проверяет, что условие ложно.
- Assert.Null(object): Проверяет, что объект равен null.
- Assert.NotNull(object): Проверяет, что объект не равен null.
- Assert.Throws<TException>(Action): Проверяет, что действие выбрасывает исключение заданного типа.
- Assert.Contains(substring, string): Проверяет, что строка содержит заданную подстроку.

3 Выводы

3.1 Протестировали программный код по методологии белого ящика;

Лабораторная работа №7
Интеграционное тестирование

1 Цель работы

1.1 Интеграционное тестирование программного кода при помощи библиотеки xUnit и Moq;

Репозиторий: https://github.com/kkerra/UnitTesting

2 Контрольные вопросы

2.1 Какой метод используется для настройки зависимостей подменяемых Moq?

Для настройки зависимостей в Moq используется метод Setup(). С помощью этого метода можно указать, какое поведение нужно симулировать для определённого метода подменяемого объекта (например, установить ожидаемое возвращаемое значение).

2.2 Какой метод используется для проверки выполнялись ли какие-либо действия в зависимостях подменяемых Moq?

Для проверки выполнения действий используются методы Verify() и VerifyAll(). Метод Verify() позволяет удостовериться, что определённый метод был вызван с заданными параметрами и заданным количеством вызовов.

2.3 Что такое интеграционное тестирование?

Интеграционное тестирование — это этап тестирования программного обеспечения, на котором проверяется взаимодействие и интеграция компонентов системы. В отличие от модульного тестирования, которое фокусируется на отдельных модулях или компонентах приложения в изоляции, интеграционное тестирование проверяет, как эти модули работают вместе.

3 Выводы

3.1 Протестировали программный код при помощи библиотеки xUnit и Moq;

Практическая работа №2
Изучение процесса формирования набора тестовых данных

1 Цель работы

1.1 Изучить основные методы и техники формирования тестовых данных для тестирования программного обеспечения;

1.2 Научиться генерировать наборы тестовых данных для различных типов тестирования.

2 Задания

2.1 Создано консольное приложение и метод, который принимает число и выводит, является ли оно простым или нет.

2.2 Граничные значения: минимальное простое число — 2, неправильные значения — 0, 1.

Эквивалентные классы: простые числа — 2, 3, 5, 7, 11. Составные числа — 4, 6, 8, 9, 10. Не числовые данные: «ааа», «5.23». Большие значения: 1000003(простое), 1000000(составное).

2.3 Реализована функция, генерирующая случайные значения в диапазоне 1 до 100.

2.4 Выполнены тесты, используя эту функцию.

2.5 Модифицирована программа, теперь она обрабатывает массив из 1000 чисел.

2.6 Сгенерирован большой набор данных, проведено нагрузочное тестирование. Время, затраченное на обработку массива из 10000 элементов — 3160 мс

Входное значение	**Ожидаемый результат**	**Класс теста**
-10	составное	Отрицательные числа
0	составное	Граничное значение
1	составное	Граничное значение
2	простое	Граничное значение(минимальное простое)
3	простое	Малые числа
4	составное	Малые составные числа
1000003	простое	Большие сложные числа
1000000	составное	Большие составные числа

3 Контрольные вопросы

3.1 Что такое набор тестовых данных?

Тестовый набор данных — это набор данных, который независим от обучающего набора данных, но который соответствует такому же распределению вероятностей, как и обучающий набор данных.

3.2 Что такое граничные значения?

Граничное значение - это минимальное (или максимальное) значение, которое находится на границе.

3.3 Что такое классы эквивалентности?

Классы эквивалентности — это набор входных (или выходных) данных программного обеспечения, которые обрабатываются программой по одному алгоритму или приводят к одному результату.

Практическая работа №3
Деление классов тестов по видам, типам и областям.

1 Цель работы

1.1 Научиться разделять тесты на различные виды и категории, а также понимать их назначение и особенности.

2 Классификация тестов

Вид теста и его описание	Пример
По уровню тестирования	
Модульное	Тестирование отдельного модуля, программы
Интеграционное	Тестирование взаимодействия между модулями
Системное	Полное тестирование всей системы
Приемочное	Тестирование на соответствие требованиям, приближенным к реальным
По видам тестирования	
Функциональное	Проверка функционала приложения
Нефункциональное	Тестирование производительности или безопасности
Регрессионное	Проверка того, что изменения не привели к сбоям
Стресс-тестирование	Проверка системы под нагрузкой, превышающей нормальные условия экспуатации
Совместимости	Проверка работы приложения на разных устройствах и браузерах
По областям	
Пользовательского интерфейса	Проверка удобства и логики взаимодействия пользователя с интерфейсом
Безопасности	Проверка уязвимостей
Производительности	Измерение времени отклика и нагрузки на приложение
Локализации	Проверка корректности перевода и адаптации интерфейса
Тестирование веб-приложений	Проверка работоспособности и безопасности веб-сайтов
Тестирование мобильных приложений	Проверка функциональности и производительности мобильных устройств

3 Выводы

3.1 Научились разделять тесты на различные виды и категории, а также понимать их назначение и особенности.

Практическая работа №4
Инспектирование кода программы.

1 Цель работы

1.1 Изучить методы инспектирования кода программы и научиться применять их на практике для повышения качества программного обеспечения.

2 Контрольные вопросы

2.1 Какие основные техники инспектирования кода существуют?

Существуют следующие основные техники инспектирования кода:

Формальная инспекция кода — формализированная процедура просмотра кода.

Неформальная инспекция кода (анализ кода) — не имеет чётких правил.

Чтение кода — самостоятельное изучение разработчиком чужого кода без присутствия автора.

Парное программирование — обзор, осуществляемый постоянно: два разработчика за одним компьютером вместе решают одну задачу.

2.2 Каковы основные критерии при инспектировании кода?

При инспектировании кода можно использовать следующие критерии:

Соответствие лучшим практикам**.**

Правильность форматирования кода**.**

Актуальность проводимых тестов и версий**.**

Реализация командой правил и политик написания кода**.**

2.3 Какие ошибки можно предотвратить при помощи инспектирования кода?

При помощи инспектирования кода можно предотвратить следующие ошибки**:**

Неправильное понимание задач.

Неправильное решение задач.

Неправильный перенос решений в код.

3 Выводы

3.1 Изучены методы инспектирования кода программы и научиться применять их на практике для повышения качества программного обеспечения.

Лабораторная работа №8
Тестирование ПО методом «черного ящика»

1 Цель работы

1.1 Освоить процесс тестирования методом «черного ящика»

2 Наборы тестов приложений

Действия	Ожидаемый результат	Полученный результат
При прохождении теста, не выбирать ответ	Не перелистывает следующую страницу с вопросом	Получилось пролистнуть весь тест не ответив не на один вопрос
Создать тесты с одинаковыми именами	Ошибка	Перезаписывает тест
В редакторе тестов выбрать тип вопроса «Один ответ»	Выбрать один вариант ответа	Можно выбрать все варианты
Изменение числа баллов за ответ	Разбалловка начинается от нуля	Можно выбрать отрицательные числа
Изменение типа вопроса во время редактирования вопроса	Изменение типа вопроса с соответствующим вариантом выбора	Тип не изменяется, для его изменения требуется пересоздавать вопрос
Видоизменение шрифта	Применение шрифта без изменений остального текста	Сбиваются абзацы
Снятие полужирного выделения	Убрать с выделенного текста полужирный	В некоторых лекциях в содержании не убирается полужирный
Удаление всех лекций	Продолжение работы приложения	Вылет приложения
Выбор открытого вопроса, задать таймер для ответа на вопроса	Функция таймера для ответа на вопрос	Данной функции нет
Создание вопроса после удаления всех вопросов в тесте	Свободное редактирование	Ошибка о выходе индекса из диапазона, тест невозможно больше редактировать и восстановить
Удаление несохраненного/несуществующего теста	Продолжение использования редактора	Вылет приложения
Изменение значения максимального балла за вопрос с помощью клавиатуры	Запрет изменения значения на большее, чем количество вопросов	Можно вводить любые значения.
Начисление баллов в редактировании вопроса	Интуитивно понятный интерфейс для начисления баллов	Не понятно, как начисляются баллы за вопрос или ответ

3 Контрольные вопросы

3.1 Что такое тестирование черного ящика?

Тестирование по стратегии чёрного ящика, часто называемое функциональным тестированием — это методика, изучающая функциональность ПО без необходимости знания внутренней структуры кода.

3.2 Какие преимущества и недостатки у тестирования черного ящика?

Преимущества тестирования «чёрного ящика»:

1. Нет необходимости анализировать внутреннюю организацию ПО.
2. Тестирование ориентировано на пользователя.
3. Раннее обнаружение проблем с интерфейсом.
4. Эффективное интеграционное тестирование.
5. Гибкость при разработке тест-кейсов.
6. Эффективность при проверке требований.
7. Подходит для крупных проектов.
8. Доступная автоматизация.
9. Простота масштабируемости.

Недостатки тестирования «чёрного ящика»:

1. Ограниченный охват кода.
2. Неспособность тестировать сложные алгоритмы.
3. Избыточное тестирование.
4. Неэффективность при выполнении повторяющихся задач.
5. Отсутствие возможности полноценно оценить производительность и масштабируемость.
6. Сложность локализации ошибок.
7. Ограничения при тестировании безопасности

3.3 Какие типы ошибок чаще всего обнаруживаются при тестировании черного ящика?

Тестирование черного ящика обеспечивает поиск следующих категорий ошибок:

некорректных или отсутствующих функций

ошибок интерфейса

ошибок во внешних структурах данных или в доступе к внешней базе данных

ошибок характеристик аппаратных устройств

ошибок инициализации и завершения

4 Выводы

4.1 Освоен процесс тестирования методом «черного ящика»

Лабораторная работа №9
Тестирование программного модуля

1 Цель работы

1.1 Освоить процесс написания сценариев тестирования

1.2 Освоить процесс тестирования по определенному сценарию

2 Набор тестов программы

Позитивные тесты:

1) корректный ввод логина, пароля и подтверждение пароля для регистрации
2) корректный ввод логина и пароля для авторизации

Негативные тесты:

1) некорректный ввод логина, пароля и подтверждение пароля при регистрации
2) заполнить не все поля ввода
3) ошибка в логине или пароле при авторизации

3 Выводы

3.1 Освоены процессы написания сценариев тестирования

3.2 Освоены процессы тестирования по определенному сценарию

Лабораторная работа №10
Отладка ПО с использованием инструментальных средств

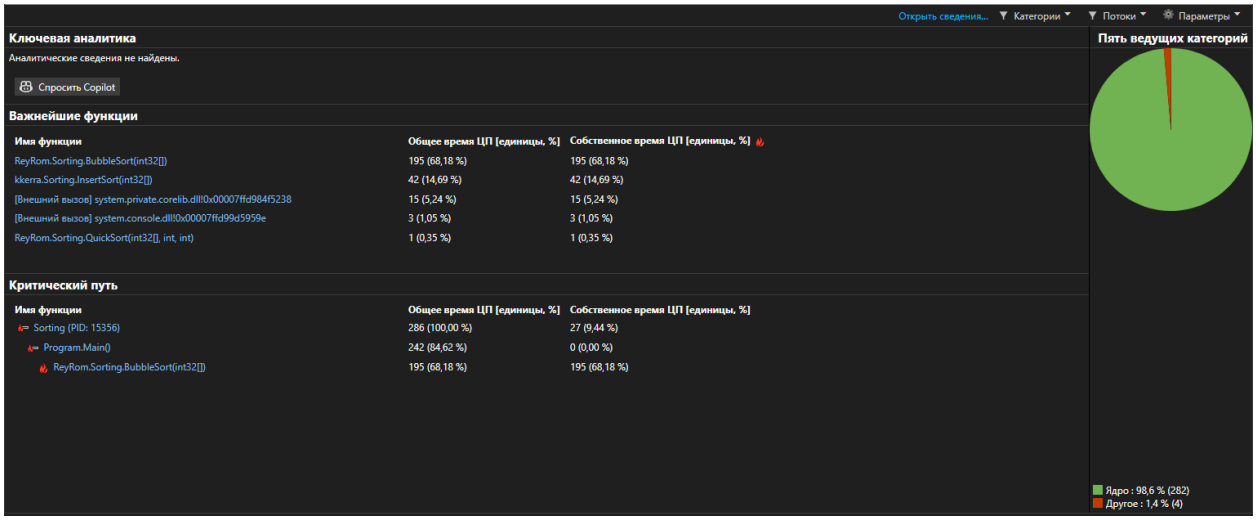
1 Цель работы

1.1 Освоить процесс отладки ПО с использованием инструментальных средств

2 Задания

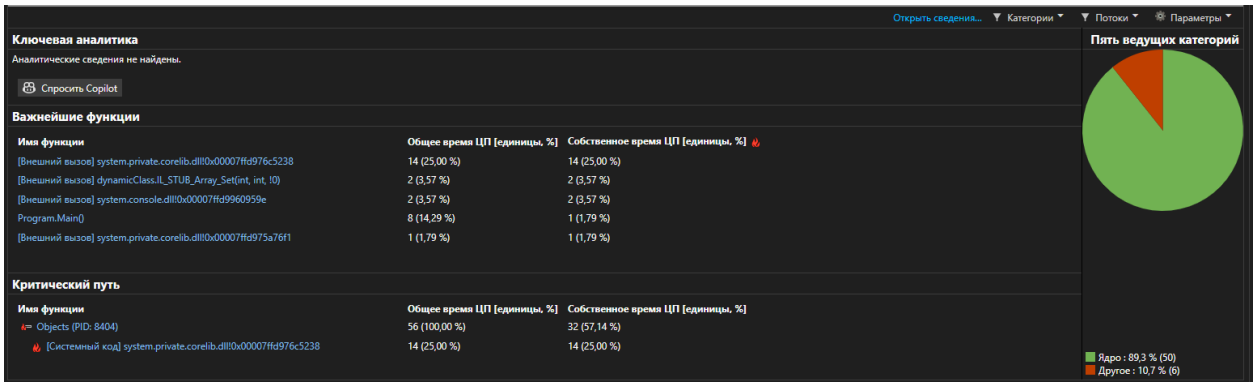
2.1 Создан форк репозитория: https://github.com/kkerra/Profiling

2.2 Запущен проект Sorting. Время, затраченное на выполнение сортировки пузырьком: 195 ms. Время, затраченное на выполнение быстрой сортировки: 1 ms. Сортировка пузырьком использовала большее время ЦП и имела больше критических путей, чем быстрая сортировка. Следовательно, сортировка пузырьком менее эффективна, чем быстрая сортировка.

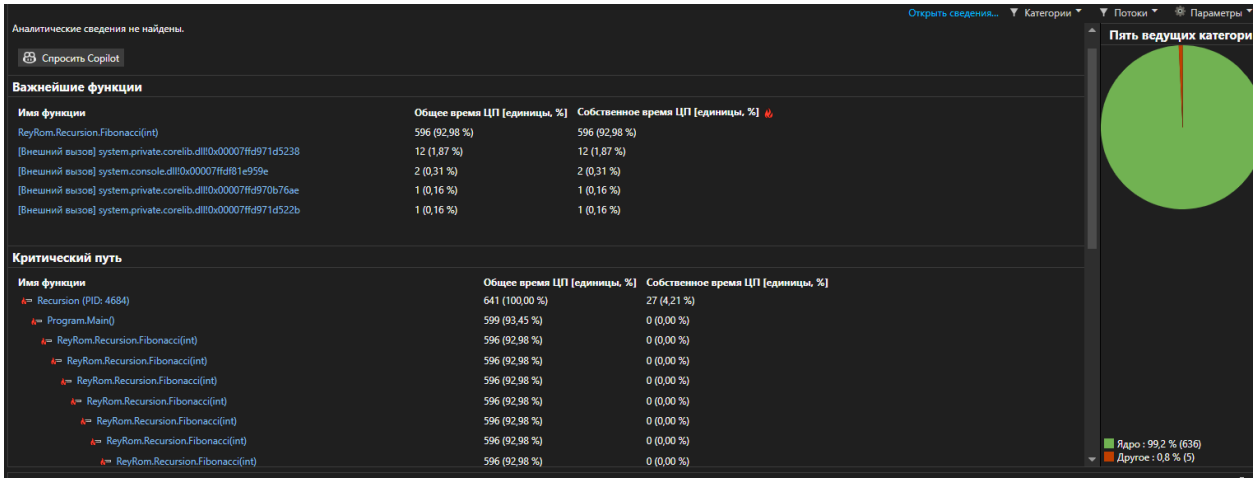


2.3 Разработана сортировка вставками. Время, затраченное на ее выполнение: 41 ms. Сортировка вставками использовала меньше ЦП, чем сортировка пузырьком, и больше, чем быстрая сортировка. Следовательно, сортировка вставками эффективнее, чем сортировка пузырьком, но менее эффективнее быстрой сортировки.

2.4 Запущен проект Objects. На создание объектов большой размерности затрачено 3 ms. На создание объектов маленькой размерности затрачено 0 ms. На создание объектов типа строк затрачено 0 ms. На создание коллекций и списков затрачено 1 ms. Метод, создающий объекты большой размерности использовал больше всего ЦП. Метод, создающий коллекции и списки использовал меньше ЦП, оставшиеся методы использовали меньше всего ЦП.



2.5 Запущен проект Recursion. Время, затраченное на выполнение: 595 ms. Функция имеет много критических путей и использует много ЦП.



2.6 Разработан рекурсивный метод, возвращающий степень числа. Время его выполнения 0 ms, использует мало ЦП.

2.7 Запущен проект MultiTasks.

3 Контрольные вопросы

4 Выводы

Лабораторная работа №13
Испытание ПО методом «Регрессионного тестирования»

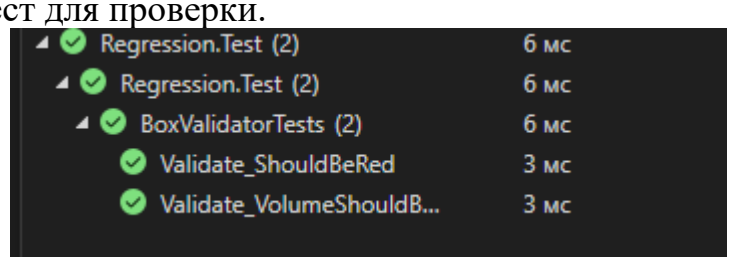
1 Цель работы

1.1 Освоить процесс применения регрессионного тестирования.

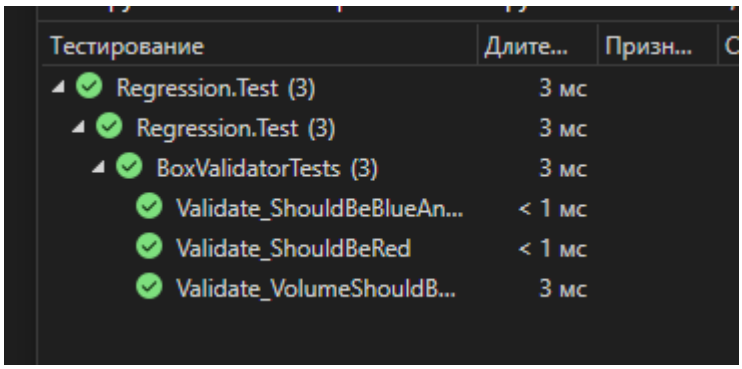
2 Порядок выполнения работы

2.1 Клонировали репозиторий

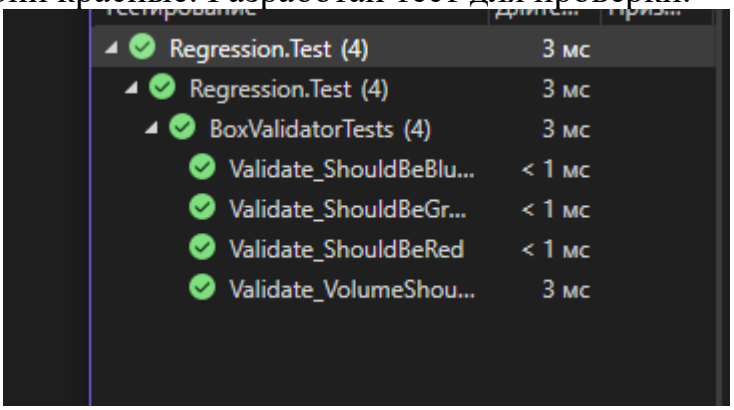
2.2 В метод Validate добавлена проверка для коробок, объемом меньше 3. Разработан тест для проверки.



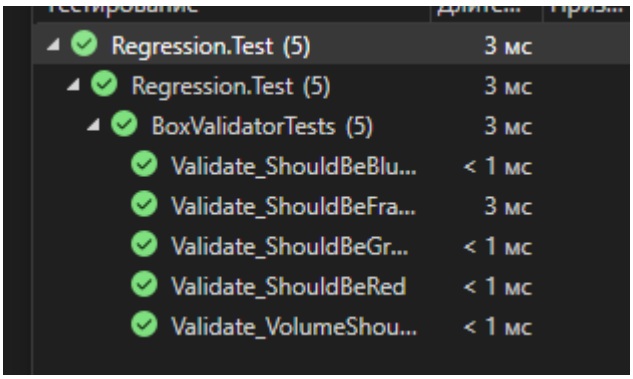
2.3 В класс Box добавлено свойство IsUrgent. В метод Validate добавлена проверка для любых синих коробок, с пометкой «Срочно». Разработан тест для проверки.



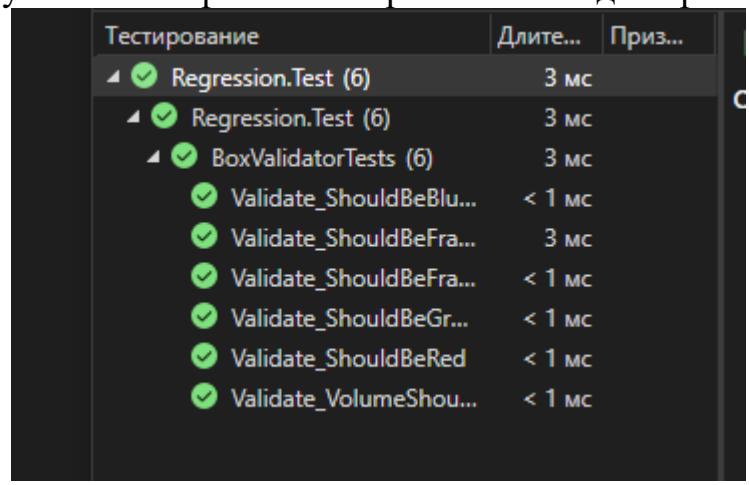
2.4 В метод Validate добавлена проверка для любых коробок, объемом больше 3, если они красные. Разработан тест для проверки.



2.5 В метод Validate добавлена проверка для любых коробок с пометкой «Хрупкое», кроме синих. Разработан тест для проверки.



2.6 В метод Validate добавлена проверка для любых коробок с пометками «Хрупкое» И «Срочно». Разработан тест для проверки.



3 Контрольные вопросы

3.1 Что такое регрессионное тестирование?

Регрессионное тестирование — это проверка ранее протестированной программы, позволяющая убедиться, что внесенные изменения не повлекли за собой появления дефектов в той части программы, которая не менялась.

4 Вывод

4.1 Освоен процесс применения регрессионного тестирования.

Лабораторная работа №11
Тестирование и отладка приложений

1 **Цель работы**

1.1 Освоить процесс разработки с использованием методологии TDD.

2 Ход работы

2.1 Реализован тест для метода AddGame.

2.2 Реализован код для прохождения тестов

2.3 Выполнен этап рефакторинга

2.4 Написаны три «Red» теста для приложения.

2.5 Отправлен запрос на слияние.

3. Контрольные вопросы

3.1 Как расшифровывается аббревиатура TTD?

"Test-Driven Development", что переводится как "разработка через тестирование". Это методология разработки программного обеспечения, при которой тесты пишутся до написания самого кода.

3.2Какие этапы обычно выделяют при разработке через тестирование?

1. Написание теста: Создание теста для новой функциональности, которая еще не реализована. Тест должен описывать ожидаемое поведение кода.

2. Запуск теста: Запуск написанного теста, который, естественно, должен завершиться неудачей, так как функциональность еще не реализована.

3. Реализация кода: Написание минимального объема кода, необходимого для того, чтобы тест прошел успешно.

4. Запуск теста повторно: Проверка, что новый код проходит тест.

5. Рефакторинг: Оптимизация и улучшение написанного кода без изменения его функциональности, при этом все тесты должны продолжать проходить.

6. Повторение процесса: Процесс повторяется для каждой новой функциональности или изменения, что способствует поддержанию высокого уровня качества кода.

4 Вывод

4.1 Освоен процесс разработки с использованием методологии TDD.