Государственное бюджетное профессиональное образовательное учреждение

Самарской области

«Тольяттинский социально-экономический колледж»

**КУРСОВАЯ РАБОТА**

**ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ «РАСЧЕТ СТОИМОСТИ УСЛУГ, ПРЕДОСТАВЛЯЕМЫХ КЛИЕНТАМ»**

**ПМ.05 «ПРОЕКТИРОВАНИЕ И разработкА ИНФОРМАЦИОННЫХ СИСТЕМ»**

**МДК 05.03 «ТЕСТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ»**

**09.02.07 Информационные системы и программирование**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент |  | **/** | Д.А. Сардов | |
|  | *подпись* |  | *И.О. Фамилия* | |
| \_\_.\_\_.2022 г. |  |  |  | |
| Оценка выполнения и защиты курсовой работы | | | | |  | |
|  | | | | |
| Руководитель |  | **/** | Н.К. Коровина | |
|  | *подпись* |  | *И.О. Фамилия* | |
| \_\_.\_\_.2022 г. |  |  |  | |

Тольятти, 2022

Государственное бюджетное профессиональное образовательное учреждение

Самарской области

«Тольяттинский социально-экономический колледж»

|  |  |  |
| --- | --- | --- |
|  |  | Утверждаю:  Заместитель директора по УР  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.С. Киронова  *« » 2022 г.* |

**ЗАДАНИЕ**

на курсовую работу

**по ПМ.05 Проектирование и разработка информационных систем модуля, выполняемой в рамках МДК.05.03 Тестирование информационных систем**

студента группы ИСП-31

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* Сардова Даниила Александровича*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия Имя отчество студента*

|  |  |
| --- | --- |
| Тема курсовой работы: **«**Расчет стоимости услуг, предоставляемых клиентам» |  |

1. Содержание задания:

1.1 Тестирование информационной системы.

1. Исходные данные:

Исходные данные для практической реализации автоматизированной информационной системы (АИС) берутся из различных информационных источников (Интернет-ресурсы, печатные издания, периодика и др.).

1. Содержание курсовой работы

Введение

1 Анализ методов тестирования

1.1 Критерии и принципы тестирования

1.2 Методы тестирования

2 Тестирование информационной системы (название)

2.1. Разработка тестовой документации (тест-дизайн)

2.2. Разработка тестовых сценариев

Заключение

Список использованных источников

Дата выдачи задания: « » 2022 г.

Дата сдачи работы на отделение: « » 2022 г.

Руководитель курсового(ой) проекта(работы) \_\_\_\_\_\_\_\_\_\_\_\_\_ / Н.К. Коровина\_\_\_\_\_\_

подпись расшифровка подписи

**КАЛЕНДАРНЫЙ ПЛАН**

выполнения курсовой работы

Студентом 3 курса группы ИСП-31

По теме Тестирование информационной системы «Расчет стоимости услуг, предоставляемых клиентам»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  этапа  работы | Содержание этапов работы | Плановый срок выполнения этапа | Планируемый объем выполнения  этапа, % | Отметка  о  выполнении  этапа |
| 1 | Выбор, обоснование темы и объекта исследования | Январь 2022 | 5% |  |
| 2 | Утверждение темы, согласование плана. Введение, библиография | Январь 2022 | 10% |  |
| 3 | Изучение и анализ информационных материалов по теме | Февраль 2022 | 15% |  |
| 4 | Обоснование актуальности выбранной темы применительно к профессиональной деятельности (введение) | Февраль 2022 | 20% |  |
| 5 | Изложение материала основной части по теме курсовой работы | Февраль 2022 | 20% |  |
| 6 | Подведение итогов проведенного анализа, формулировка выводов УИР применительно к профессиональной деятельности (заключение) | Март 2022 | 20% |  |
| 7 | Оформление работы и сдача на проверку | Март 2022 | 10% |  |
| 8 | Защита работы |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  | **/** | Д.А. Сардов |
|  | *подпись* |  | *И.О. Фамилия* |
| 17.01.2022 г. |  |  |  |
| Руководитель |  | **/** | Н.К. Коровина |
|  | *подпись* |  | *И.О. Фамилия* |
| 17.01.2022 г. |  |  |  |

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc101374556)

[1 АНАЛИЗ МЕТОДОВ ТЕСТИРОВАНИЯ 7](#_Toc101374557)

[1.1 Критерии и принципы тестирования 7](#_Toc101374558)

[1.2 Методы тестирования 11](#_Toc101374559)

[2 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ РАСЧЕТ СТОИМОСТИ УСЛУГ, ПРЕДОСТАВЛЯЕМЫХ КЛИЕНТАМ 14](#_Toc101374560)

[2.1. Разработка тестовой документации (тест-дизайн) 14](#_Toc101374561)

[2.2. Разработка тестовых сценариев 17](#_Toc101374562)

[ЗАКЛЮЧЕНИЕ 23](#_Toc101374563)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 24](#_Toc101374564)

# ВВЕДЕНИЕ

Известно, что основной задачей первых трех десятилетий компьютерной эры являлось развитие аппаратных компьютерных средств. Это было обусловлено высокой стоимостью обработки и хранения данных. В 80-е годы успехи микроэлектроники привели к резкому увеличению производительности компьютера при значительном снижении стоимости.

Основной задачей 90-х годов XX и начала XXI века стало совершенствование качества компьютерных приложений, возможности которых целиком определяются программным обеспечением (ПО).

Современный персональный компьютер имеет производительность большой ЭВМ 80-х годов. Сняты практически все аппаратные ограничения на решение задач. Оставшиеся ограничения приходятся на долю ПО.

При изучении предметной области, выяснилось, что информационная система «Расчет стоимости услуг, предоставляемых клиентам» на первый взгляд простая, но для ее реализации необходимо обратится к некоторым разделам, таким как создание базы данных и создание приложения для взаимодействия с этой базой данных.

Объектом исследования курсовой работы является процесс расчета стоимости услуг, предоставляемых клиенту на выбор, в студии автозвука «Sound Studio».

Предметом исследования процесса будет ИС «Расчет стоимости услуг, предоставляемых клиенту».

Выше изложенное в целом на теоретико-методологическом уровне определило проблему настоящего исследования: тестирование программного модуля «Расчет стоимости» в программе MS Visual Studio 2019.

Целью курсовой работы является изучить процесс тестирования как составляющую процесса обеспечения качества разработки ПО, а также теоретически обосновать основные положения данного процесса и проверить их практически на основе разработанной информационной системы «Расчет стоимости услуг, предоставляемых клиенту».

Для достижения поставленной цели были определены следующие задачи:

# проанализировать методы тестирования;

# изучить критерии и принципы тестирования;

# изучить методы тестирования;

# протестировать информационную систему;

# разработать тестовую документацию;

# разработать тестовые сценарии.

Для тестирования информационной системы будут применяться методы: тестирование деятельности предприятия с выявлением его функции, а также функциональное тестирование системы.

Практическая значимость заключается в использовании протестированной системы на различных предприятиях по «Расчету стоимости услуг».

Структура работы соответствует логике исследования и включает в себя введение, две главы, заключение, список использованной литературы.

# 1 АНАЛИЗ МЕТОДОВ ТЕСТИРОВАНИЯ

## 1.1 Критерии и принципы тестирования

Можно выделить требования к идеальному критерию тестирования:

# Критерий должен быть достаточным, т.е. показывать, когда некоторое конечное множество тестов достаточно для тестирования данной программы;

# Критерий должен быть полным, т.е. в случае ошибки должен существовать тест из множества тестов, удовлетворяющих критерию, который раскрывает ошибку;

# Критерий должен быть надежным, т.е. любые два множества тестов, удовлетворяющих ему, одновременно должны раскрывать или не раскрывать ошибки программы;

# Критерий должен быть легко проверяемым, например, вычисляемым на тестах.

Для нетривиальных классов программ в общем случае не существует полного и надежного критерия, зависящего от программ или спецификаций. Поэтому, как правило, стремятся к идеальному общему критерию через реальные частные.

Следом рассмотрим классы критериев тестирования:

# Структурные критерии используют информацию о структуре программы (критерии так называемого «белого ящика»);

# Функциональные критерии формулируются в описании требований к программному изделию (критерии так называемого «черного ящика»);

# Критерии стохастического тестирования формулируются в терминах проверки наличия заданных свойств у тестируемого приложения, средствами проверки некоторой статистической теории;

# Мутационные критерии ориентированы на проверку свойств программного изделия на основе подхода Монте-Карло.

Рассмотрим каждый критерий отдельно.

Структурные критерии используют модель программы в виде «белого ящика», что предполагает знание исходного текста программы или спецификации программы в виде потокового графа управления. Структурная информация понятна и доступна разработчикам подсистем и модулей приложения, поэтому данный класс критериев часто используется на этапах модульного и интеграционного тестирования.

Структурные критерии базируются на основных элементах УГП, операторах, ветвях и путях.

Условие критерия тестирования команд (критерий С0) – набор тестов в совокупности должен обеспечить прохождение каждой команды не менее одного раза. Это слабый критерий, используется в больших программных системах, где другие критерии применить невозможно.

Условие критерия тестирования ветвей (критерий С1) – набор тестов в совокупности должен обеспечить прохождение каждой ветви не менее одного раза. Это достаточно сильный и при этом экономичный критерий. Данный критерий часто используется в системах автоматизации тестирования.

Условие критерия тестирования путей (критерий С2) – набор тестов в совокупности должен обеспечить прохождение каждого пути не менее одного раза. Если программа содержит цикл, то число итераций ограничивается константой (часто – 2, или числом классов выходных путей).

Структурные критерии не проверяют соответствие спецификации, если

оно не отражено в структуре программы.

Функциональный критерий – важнейший для программной индустрии критерий тестирования. Он обеспечивает, прежде всего, контроль степени выполнения требований заказчика в программном продукте. Поскольку требования формулируются к продукту в целом, они отражают взаимодействие тестируемого приложения с окружением. При функциональном тестировании преимущественно используется модель «черного ящика». Проблема функционального тестирования – это, прежде всего, трудоемкость.

Стохастическое тестирование применяется при тестировании сложных программных комплексов - когда набор детерминированных тестов (X, Y) имеет громадную мощность. В случаях, когда подобный набор невозможно разработать и исполнить на фазе тестирования, можно применить следующую методику.

Постулируется, что профессиональные программисты пишут сразу почти правильные программы, отличающиеся от правильных мелкими ошибками или описками типа – перестановка местами максимальных значений индексов в описании массивов, ошибки в знаках арифметических операций, занижение или завышение границы цикла на 1 и т.п. Предлагается подход, позволяющий на основе мелких ошибок оценить общее число ошибок, оставшихся в программе.

Подход базируется на следующих понятиях:

# Мутации – мелкие ошибки в программе;

# Мутанты – программы, отличающиеся друг от друга мутациями.

Метод мутационного тестирования – в разрабатываемую программу P вносят мутации, т.е. искусственно создают программы-мутанты P1, P2…Затем программа P и ее мутанты тестируются на одном и том же наборе тестов {X,Y}.

Если на наборе {X,Y} подтверждается правильность программы P и, кроме того, выделяются все внесенные в программы-мутанты ошибки, то набор тестов (X,Y) соответствует мутационному критерию, а тестируемая программа объявляется правильной.

Если некоторые мутанты не выявили всех мутаций, то надо расширять набор тестов (X,Y) и продолжать тестирование.

После вышеописанного, проанализируем 7 принципов тестирования.

Эти принципы тестирования были предложены в последние 40 лет и являются общим руководством для тестирования в целом.

Первый принцип – тестирование демонстрирует наличие дефектов.

Тестирование может показать, что дефекты присутствуют, но не может доказать, что их нет.

Тестирование снижает вероятность наличия дефектов, находящихся в программном обеспечении, но, даже если дефекты не были обнаружены, это не доказывает его корректности.

Второй принцип – исчерпывающее тестирование недостижимо.

Полное тестирование с использованием всех комбинаций вводов и предусловий физически невыполнимо, за исключением тривиальных случаев. Вместо исчерпывающего тестирования должны использоваться анализ рисков и расстановка приоритетов, чтобы более точно сфокусировать усилия по тестированию.

Третий принцип – раннее тестирование.

Чтобы найти дефекты как можно раньше, действия по тестированию должны быть начаты как можно раньше в жизненном цикле разработки программного обеспечения или системы, и должны быть сфокусированы на определенных целях.

Четвертый принцип – скопление дефектов.

Усилия тестирования должны быть сосредоточены пропорционально ожидаемой, а позже реальной плотности дефектов по модулям. Как правило, большая часть дефектов, обнаруженных при тестировании или повлекших за собой основное количество сбоев системы, содержится в небольшом количестве модулей.

Пятый принцип – парадокс пестицида.

Если одни и те же тесты будут прогоняться много раз, в конечном счете этот набор тестовых сценариев больше не будет находить новых дефектов. Чтобы преодолеть этот «парадокс пестицида», тестовые сценарии должны регулярно рецензироваться и корректироваться, новые тесты должны быть разносторонними, чтобы охватить все компоненты программного обеспечения, или системы, и найти как можно больше дефектов.

Шестой принцип – тестирование зависит от контекста.

Тестирование выполняется по-разному в зависимости от контекста. Например, программное обеспечение, в котором критически важна безопасность, тестируется иначе, чем сайт электронной коммерции.

Седьмой принцип – заблуждение об отсутствии ошибок.

Обнаружение и исправление дефектов не помогут, если созданная система не подходит пользователю и не удовлетворяет его ожиданиям и потребностям.

## 1.2 Методы тестирования

Существуют различные методы, которые можно использовать для тестирования программного обеспечения.

Методика тестирования без каких-либо знаний о внутренней работе приложения называется «черным ящиком». Тестер не обращает внимания на архитектуру системы и не имеет доступа к исходному коду. Как правило, при выполнении теста с «черным ящиком» тестер будет взаимодействовать с пользовательским интерфейсом системы, предоставляя входные данные и анализируя выходы, не зная, как и где обрабатываются входы.

Преимущества данного метода следующие:

# Хорошо подходит и эффективен для больших сегментов кода;

# Кодовый доступ не требуется;

# Четкое разделение перспективы пользователя с точки зрения разработчика с помощью явно определенных ролей;

# Большое количество умеренно квалифицированных тестировщиков может протестировать приложение без каких-либо знаний о реализации, языке программирования или операционных системах.

Недостатки метода:

# Ограниченное покрытие, поскольку на самом деле выполняется только выбранное количество тестовых сценариев;

# Неэффективное тестирование, из-за того, что тестер только имеет ограниченные знания о приложении;

# Слепой охват, поскольку тестер не может ориентироваться на определенные сегменты кода или области ошибок;

# Тестовые примеры трудно разработать.

Рассмотрим тестирование «белого ящика».

Проверка белого ящика – это подробное исследование внутренней логики и структуры кода. Тестирование с использованием белого ящика также называется тестированием стекла или открытым тестированием. Чтобы выполнить тестирование белого ящика в приложении, тестер должен знать внутреннюю работу кода.

Тестер должен заглянуть внутрь исходного кода и выяснить, какое устройство/блок кода ведет себя некорректно.

Преимущества данного метода:

# Поскольку тестер знает исходный код, становится очень легко узнать, какой тип данных может помочь в эффективном тестировании приложения;

# Это помогает в оптимизации кода;

# Дополнительные строки кода могут быть удалены, что может привести к скрытым дефектам;

# Благодаря знаниям тестера о коде, максимальный охват достигается при написании сценария.

Недостатки метода:

# В связи с тем, что для тестирования белых ящиков требуется квалифицированный тестер, затраты увеличиваются;

# Иногда невозможно заглянуть в каждый уголок и угол, чтобы обнаружить скрытые ошибки, которые могут создавать проблемы, так как многие пути будут не проверены;

# Трудно поддерживать тестирование белых ящиков, поскольку для этого требуются специализированные инструменты, такие как анализаторы кода и инструменты отладки.

Изучим тестирование серых ящиков.

Тестирование на серой коробке – это метод тестирования приложения с ограниченным знанием внутренней работы приложения. При тестировании программного обеспечения фраза, чем больше вы знаете, тем лучше переносит массу при тестировании приложения.

Освоение домена системы всегда дает тестеру преимущество над кем-то с ограниченными знаниями домена. В отличие от тестирования черного ящика, где тестер тестирует только пользовательский интерфейс приложения; при тестировании в сером полете тестер имеет доступ к проектной документации и базе данных.

Преимущества данного метода:

# Предлагает комбинированные преимущества тестирования черного ящика и белого ящика, где это возможно;

# Тестировщики серого ящика не полагаются на исходный код; вместо этого они полагаются на определение интерфейса и функциональные спецификации;

# Основываясь на имеющейся ограниченной информации, тестер серого ящика может разработать отличные сценарии тестирования, особенно в отношении протоколов связи и обработки данных;

# Тест выполняется с точки зрения пользователя, а не дизайнера.

Недостатки метода:

# Поскольку доступ к исходному коду недоступен, возможность пройти через код и зону тестирования ограничена;

# Тесты могут быть излишними, если разработчик программного обеспечения уже выполнил тестовый пример;

# Тестирование всех возможных входных потоков нереально, поскольку для этого потребуется необоснованное количество времени; поэтому многие программные пути будут не проверены.

# 2 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ РАСЧЕТ СТОИМОСТИ УСЛУГ, ПРЕДОСТАВЛЯЕМЫХ КЛИЕНТАМ

## 2.1. Разработка тестовой документации (тест-дизайн)

Документ описывает методы и подходы к тестированию, которые будут использоваться тестировщиками отдела тестирования, для тестирования приложения. План тестирования может использоваться как тестировщиками, так и менеджерами, разработчиками. Объект тестирования — это деятельность, направленная на проверку работоспособности функций ИС «Расчет стоимости услуг, предоставляемых клиентам» приложения «Sound Studio».

Целью тестирования приложения «Sound Studio» является проверка корректной работы.

Итогом процесса тестирования будут следующие материалы:

1. Определить существующую информацию о проекте и программных компонентах, подлежащих тестированию.
2. Описать стратегии тестирования, которые будут использоваться.
3. Определить необходимые ресурсы для проведения тестирования.
4. Привести результаты тестирования.

Результаты будут отправлены заказчику в виде отчетов.

Условия для тестирования. Приложение должно удовлетворять потребность пользователя в активностях, связанных с просмотром каталога услуг, делать заказы, пользоваться обратной связью, рассчитывать стоимость.

Стратегия процесса тестирования. Приведенный ниже план тестирования является формальным, так как для построения развернутого плана необходимо понимание текущего состояния проекта.

Основными задачами тестирования являются:

* проведение функционального тестирования каждого модуля и компонента системы для обеспечения его соответствия функциональным требованиям;
* тестирование данных и целостности базы данных.

Виды тестирования. Для решения указанных выше задач тестирования будут использоваться следующие виды тестирования.

1. Тестирование данных и целостности базы данных

Базы данных должны тестироваться как отдельные системы внутри ИС «Расчет стоимости услуг, предоставляемых клиентам». Эти системы должны тестироваться отдельно от приложений (таких как интерфейс доступа к данным).

Необходимо провести дополнительное исследование СУБД на тему того, какие инструменты/техники существуют для выполнения нижеописанного тестирования.

1.1 Цель тестирования

Убедится в том, что методы доступа к данным работают правильно и без нарушения целостности БД.

1.2 Способы

* Вызвать каждый метод доступа к БД, предоставляя правильные и не правильные данные (или запросы к данным).
* Исследовать БД на предмет корректного заполнения ее данными, корректной обработки событий

1.3 Критерий завершенности

Все методы и процедуры БД функционируют так, как им положено и без нарушения целостности самой БД.

1.4 Особые замечания

* При тестировании может понадобиться среда разработки СУБД или драйвера для корректного подключения к базам данных.
* Процедуры должны вызываться вручную.
* Для повышения видимости неприемлемых событий БД необходимо использовать небольшие БД или БД с ограниченным количеством записей.

2. Функциональное тестирование

1.1 Цель тестирования

Функциональное тестирование состоит в том, чтобы убедиться, что весь программный продукт работает в соответствии с требованиями, и в приложении не появляется существенных ошибок.

1.2 Способы

Авторизация:

Авторизация пользователя

Восстановление пароля

Редактирование учетной записи

Оформление заказа:

Возврат в шагах при оформлении

Расчет стоимости заказа

Изменение/удаление количества товаров на этапе заказа

Отправка и доставка сообщений

1.3 Критерий завершенности

Программный продукт должен пройти все запланированные тесты.

1.4 Особые замечания

Ожидаемые результаты возникают при использовании достоверных данных.

Соответствующие сообщения об ошибках или предупреждения отображаются, когда используются неверные данные.

Подготовлено тестовое окружение, приложение готово к тестированию на тестовой площадке.

Не будет проведено нагрузочное и тестирование безопасности в виду отсутствия необходимых ресурсов.

Отчеты об ошибках создаются для того, чтобы предоставить команде разработчиков и руководителю проекта исчерпывающую информацию об обнаруженных ошибках. Они должны быть полезны при определении причин ошибок и их исправлении.

Продукт должен работать в соответствии с требованиями и техническим заданием. Продукт не должен содержать критических и блокирующих дефектов в окончательной версии проекта.

## 2.2. Разработка тестовых сценариев

Unit test – блок кода (обычно метод), который вызывает тестируемый блок кода и проверяет его правильность работы. Если результат Unit – теста не совпадает с ожидаемым результатом, тест считается не пройденным.

Модульное тестирование (Unit testing) – тестирование каждой атомарной функции приложения отдельно, с использованием объектов искусственно смоделированной среды.

Создадим новый проект – библиотека классов (Class Library), и назовем его MyCalcLib (рисунок 1).

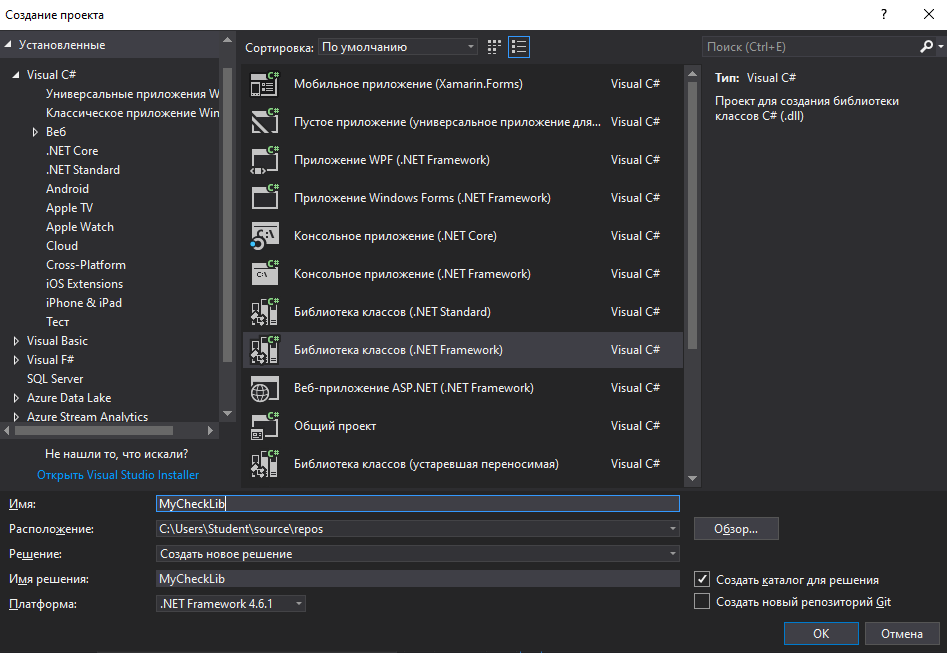


Рисунок 1 – Создание проекта

Переименуем созданный класс из Class1 в MyCheck.

Далее добавим открытый метод, который будет возвращать string, с названием Sum в качестве параметра он принимает string fname x и string sname, и этом методе возвращаем fname + sname. Текст кода предоставлен на рисунке ниже:

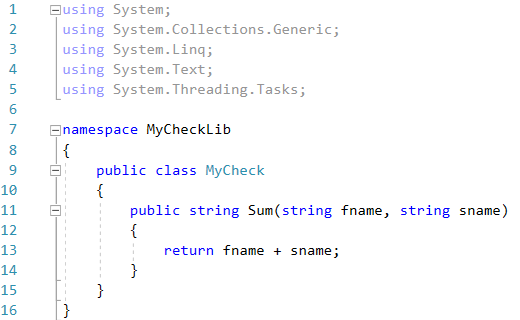


Рисунок 2 – Код метода

Для этого метода нам необходимо создать Unit Test. Правой кнопкой

мыши кликаем в обозреватели решений по решению MyCheckLib, выбираем

добавить новый проект (рисунок 3). Затем в списке проектов переключаемся на вкладку Test, далее в правой части необходимо указать тип теста: модульный тест, веб-тест или тест пользовательского интерфейса, в данном случае выбираем проект модульного теста (Unit Test Project) и назовем его MyCheckLib.Tests (рисунок 4).

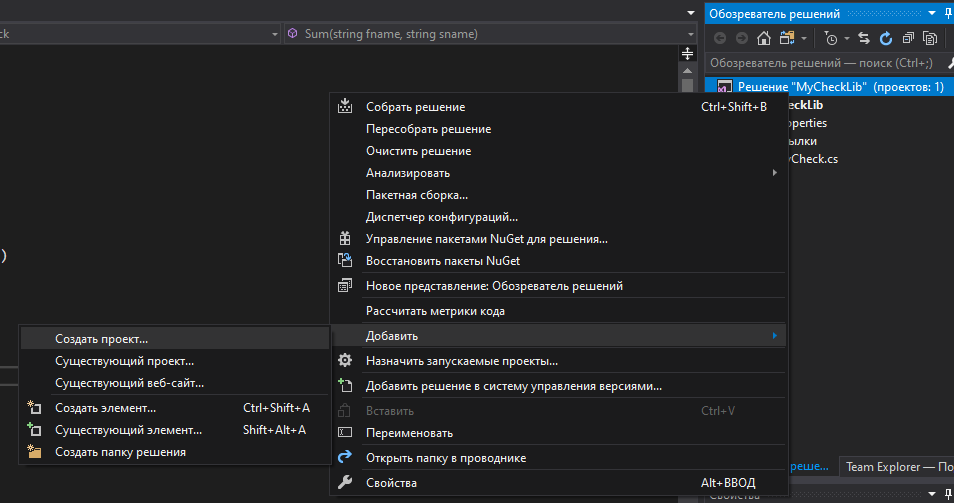


Рисунок 3 – Добавление нового проекта

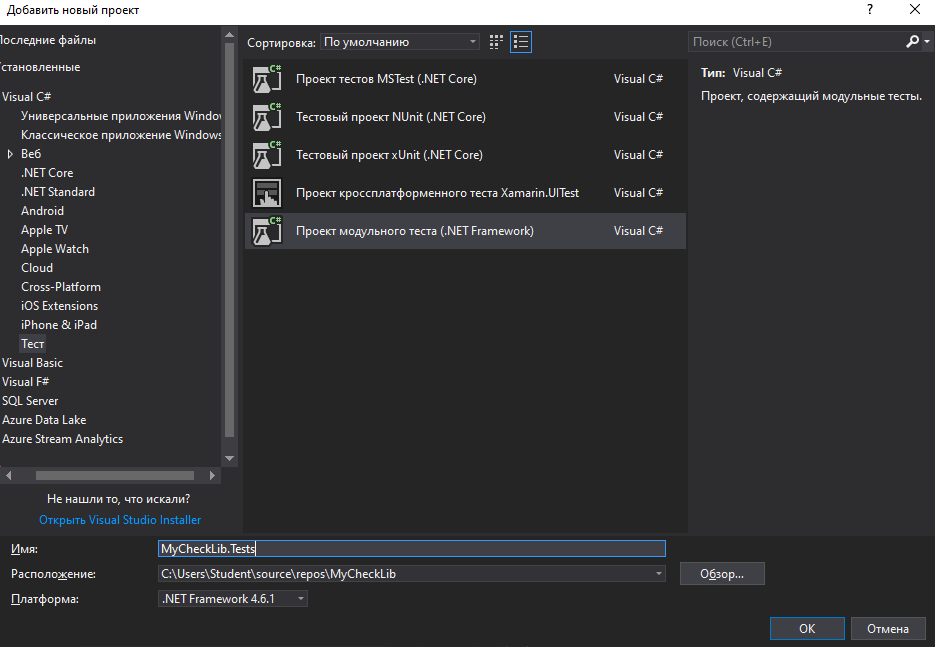


Рисунок 4 – Окно добавление нового проекта

Переименуем в обозреватели решений UnitTest1 в MyCheckTests.

Unit – тестами являются методы, которые помечены атрибутом [TestMethod], Unit – тесты собираются в классы, которые помечены атрибутом [TestClass]. Текст кода предоставлен на рисунке ниже:

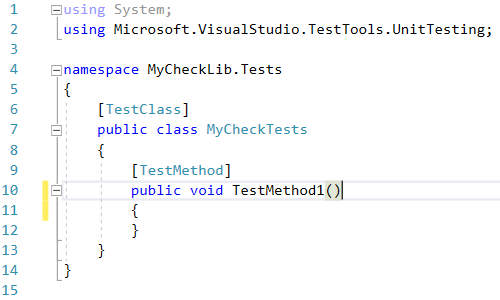


Рисунок 5 – Текст кода

Чтобы использовать TestClass и TestMethod атрибуты необходимо подключить пространство имен Microsoft.VisualStudio.TestTools.UnitTesting.

Для того чтобы мы имели возможность работать с классом MyCheck.cs, который находится в отдельной сборке, необходимо эту сборку подключить, для этого правой кнопкой мыши кликаем в обозреватели решений по папке References в MyCheckLib.Tests, и в контекстном меню выбираем добавить ссылку.

Выбираем проект MyCheckLib из текущего решения. Теперь эта сборка подключена.

Теперь необходимо написать Unit – тест, который будет проверять правильность работы метода Sum, для этого вначале переименуем имя метода из TestMethod1 в Sum\_fnameANDsname\_expectedreturned (это метод на вход будет принимать два значения fname и sname, результат, который будет возвращен этим методом будет равен expected), это необходимо для того чтобы четко понимать и отображать что именно мы тестируем. Далее мы делаем три операции arrange (здесь создаются переменные), act (здесь выполняются действия над системой) и assert (проверка, что операции выполнились успешно и Unit – тест завершен правильно). Текст кода представлен на рисунке ниже:

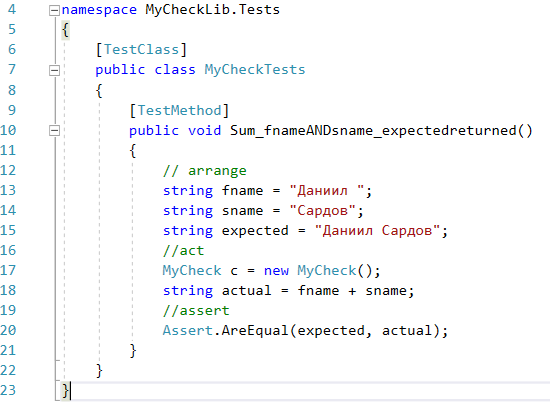


Рисунок 6 – Текст кода

Если в результате выполнения фрагмента кода приведенного выше значения expected и actual будут равны, то в таком случае мы получим результат true и Unit – тест – закончится успешно, если же они не будут равны, то мы получим ошибку.

Для запуска теста в обозреватели теста (Test Explorer), для того чтобы вызвать окно обозревателя решений кликаем ТЕСТ – окна – обозреватель тестов, кликом по необходимому тесту вызвать контекстное меню и в контекстном меню кликнуть по выполнить выбранные тесты или же в верхней панели в закладке тест пройти по вкладке выполнить и в нем выбрать необходимый вариант. Если тесты отсутствуют в обозреватели решений, необходимо выполнить сборку решения, для этого в верхней панели выбираем СБОРКА (Build) – Собрать решение.

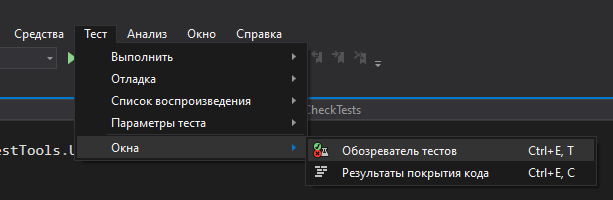


Рисунок 7 – Включение окна обозреватель тестов

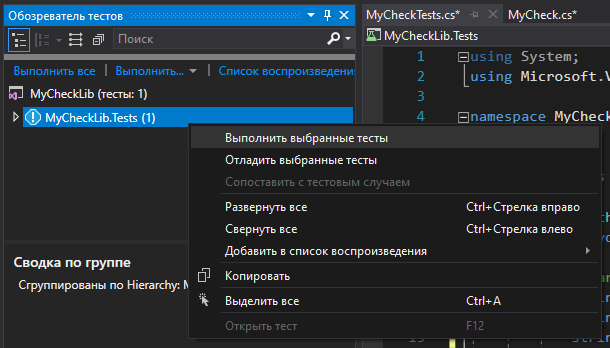


Рисунок 8 – Вызов теста из окна обозреватель тестов

Когда тест завершился если он завершился удачно рядом с тестом должна появиться зеленая иконка и также в нижней части обозревателя теста не должно выводиться не каких ошибок. Также после успешного завершения теста можно посмотреть время его исполнения (в нашем случае это 51 мс).

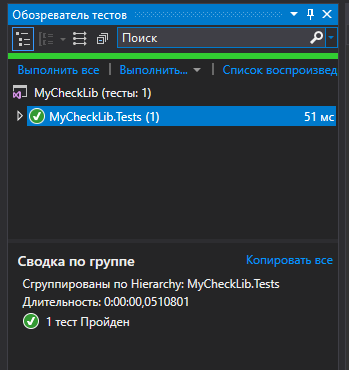


Рисунок 9 – Результат теста

Если мы выполним изменения в методе Sum, и они приведут к тому, что появятся ошибки, то после запуска Unit – теста, мы увидим какая произошла ошибка и в каком методе. Красная иконка свидетельствует о том, что в тесте произошел сбой.

# ЗАКЛЮЧЕНИЕ

В первой части курсовой работы были проанализированы методы тестирования. Тестирование – это процесс анализа пункта требований к ПО с целью фиксации различий между существующим состоянием ПО и требуемым (что свидетельствует о проявлении ошибки) при экспериментальной проверке соответствующего пункта требований. Были изучены критерии и принципы тестирования информационной системы. Также, были описаны методы тестирования ИС, такие как метод «черного ящика», метод «белого ящика», метод «серого ящика».

Во второй части курсовой работы на основе изученных методов тестирования была разработана и подробно описана тестовая документация (тест-дизайн) информационной системы, а также был разработан тестовый сценарий для информационной системы «Расчет стоимости услуг, предоставляемых клиентам». В целом это составляет тестирование ИС.

Практической частью курсовой работы было тестирование фрагмента информационной системы «Расчет стоимости услуг, предоставляемых клиентам».

Подробно описаны и проиллюстрированы функционал и структура самого модуля, а также проведено тестирование готового программного продукта в программе MS Visual Studio 2019. В последнюю очередь была составлена сопутствующая документация.

Таким образом, задачи, сформулированные во введении, решены, а цель достигнута.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 34.601 – 90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания.

2. ГОСТ 34.602 - 2020. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.

3. ГОСТ 19.201 - 78 ЕСПД. Техническое задание. Требования к содержанию и оформлению.

4. ГОСТ 19.202 - 78 ЕСПД. Спецификация. Требования к содержанию и оформлению.

5. ГОСТ Р ИСО/МЭК 12207. Процессы жизненного цикла программных средств.

6. Золотов С.Ю. Проектирование информационных систем: Учебно-методическое пособие. Методические рекомендации для выполнения курсового проекта, лабораторных работ и практических занятий по дисциплине «Проектирование информационных систем» - Томск: ТУСУР, 2013. - 34 с.

7. Шнайдер, Роберт Microsoft SQL Server 6.5. Проектирование высокопроизводительных баз данных; М.: Лори, 2010. - 361 c

8. Петкович, Душан Microsoft SQL Server 2012. Руководство для начинающих / Душан Петкович. - М.: БХВ-Петербург, 2012. - 460 c.

9. Тейлор, Аллен SQL для чайников / Аллен Тейлор. - М.: Вильямс, 2014. - 416 c.

10. Браст, Э.Дж. Разработка приложений на основе Microsoft SQL Server 2008 / Э.Дж. Браст. - М.: Русская Редакция, 2010. - 751 c.

11. Хетагуров, Я. А. Проектирование автоматизированных систем обработки информации и управления (АСОИУ). Учебник / Я.А. Хетагуров. - М.: Бином. Лаборатория знаний, 2015. - 240 c.

12. Кристофер, Д. Маннинг Введение в информационный поиск / Кристофер Д. Маннинг, ПрабхакарРагхаван ,ХайнрихШютце. - М.: Вильямс, 2014. - 528 c.

13. Бишоп Дж. С# в кратком изложении; Бином. Лаборатория знаний - М., 2015. - 234 c.

14. ГриффитсИэн Программирование на C# 5.0; Эксмо - М., 2014. - 580 c.

15. Гуриков С. Р. Введение в программирование на языке Visual C#; ИЛ - Москва, 2013. - 448 c.

16. Шилдт Герберт C# 4.0. Полное руководство; Вильямс - М., 2015. - 291 c.

17. Эндрю Троелсен Язык программирования C# 5.0 и платформа .NET 4.5; Диалектика / Вильямс - М., 2015. - 126 c.