Государственное бюджетное профессиональное образовательное учреждение

Самарской области

«Тольяттинский социально-экономический колледж»

**КУРСОВАЯ РАБОТА**

**ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ «УЧЁТ ОПЛАТЫ ЗА ОБУЧЕНИЕ В ХУДОЖЕСТВЕННОЙ ШКОЛЕ»**

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

**МДК 05.03 «ТЕСТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ»**

**09.02.07 «ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММИРОВАНИЕ»**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  | **/** | О.А. Федорова |
|  | *подпись* |  | *И.О. Фамилия* |
| 02.09.2021 г. |  |  |  |
| Руководитель |  | **/** | *Н.К. Коровина* |
|  | *подпись* |  | *И.О. Фамилия* |
| 02.09.2021 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-32

Федорова Ольга Андреевна

*Фамилия Имя отчество студента*

|  |  |
| --- | --- |
| Тема курсовой работы: | «Учёт оплаты за обучение в художественной школе» |

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 (Компьютерный магазин)

2.1. Разработка тестовой документации (тест-дизайн)

2.2. Разработка тестовых сценариев

Заключение

Список использованных источников

Дата выдачи задания: « » 202 г.

Дата сдачи работы на отделение: « » 202 г.

Руководитель курсового(ой) проекта(работы) \_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись расшифровка подписи

Государственное бюджетное профессиональное образовательное учреждение

Самарской области

«Тольяттинский социально-экономический колледж»

**КАЛЕНДАРНЫЙ ПЛАН**

выполнения курсовой работы

Студентом \_курса 3 группы ИСП -32

Федорова О.А

*Фамилия, И.О.*

По теме

Компьютерный магазин

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  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 |  | **/** | Федорова О. А. |
|  | *подпись* |  | *И.О. Фамилия* |
| 02.09.2021 г. |  |  |  |
| Руководитель |  | **/** | *Н.К. Коровина* |
|  | *подпись* |  | *И.О. Фамилия* |
| 02.09.2021 г. |  |  |  |

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc101620556)

[1 АНАЛИЗ МЕТОДОВ ТЕСТИРОВАНИЯ 6](#_Toc101620560)

[1.1 Критерии и принципы тестирования 6](#_Toc101620561)

[1.2 Методы тестирования 10](#_Toc101620562)

[2 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ (Учёт оплаты) 13](#_Toc101620563)

[2.1. Разработка тестовой документации (тест-дизайн) 13](#_Toc101620564)

[2.2. Разработка тестовых сценариев 16](#_Toc101620565)

[ЗАКЛЮЧЕНИЕ 22](#_Toc101620566)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](#_Toc101620567)

# ВВЕДЕНИЕ

# Каждый день в своей работе мы сталкиваемся с достаточно абстрактным понятием «качество ПО» и если задать вопрос тестировщику или программисту «что такое качество?», то у каждого найдется своё толкование. Рассмотрим определение "качества ПО" в контексте международных стандартов:

# Качество программного обеспечения — это степень, в которой ПО обладает требуемой комбинацией свойств.

# Качество программного обеспечения — это совокупность характеристик ПО, относящихся к его способности удовлетворять установленные и предполагаемые потребности

В проектах по разработке программного обеспечения (ПО) помимо основной задачи по реализации заявленной функциональности

существует не менее важная задача по обеспечению качества ПО.

Качество ПО (Software quality) — это совокупность характеристик

программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности. Каждый участник проекта может иметь собственное представление о критериях качества и оценке степени присутствия критериев качества в ПО. Таким

образом, возникает задача определения круга заинтересованных лиц,

согласования набора критериев качества и нахождения оптимального

баланса критериев качества, обеспечивающего качественное ПО.

Структура работы **–** этозавершающая часть введения(что в итоге в работе представлено?).

В завершающей части в назывном порядке перечисляются структурные части работы, например: «Структура работы соответствует логике исследования и включает в себя введение, три главы, восемь параграфов, заключение, список источников и литературы, приложения».

Выше изложенное в целом на теоретико-методологическом уровне определило проблему настоящего исследования: разработка программного модуля «Компьютерный магазин»

1. Анализ предметной области.
2. Разработка функциональной модели.
3. Разработка логической и физической структуры.
4. Разработка информационной системы «Компьютерный магазин».
5. Разработка интерфейса ИС.
6. Разработка руководства пользователя.
7. Тестирование разработанного ПП.

Методы исследования: для разработки информационной системы будут применяться методы анализ деятельности предприятия с выявлением его функции, а также функциональное моделирование системы.

# 1 АНАЛИЗ МЕТОДОВ ТЕСТИРОВАНИЯ

## Критерии и принципы тестирования

Тестирование как метод верификации — это парадокс. Тестирование программы для того, чтобы проверить ее качество, теоретически равносильно втыканию булавок в куклу, причем очень маленьких булавок в очень большую куклу. Разрешить этот парадокс можно, определив реалистичные ожидания.

Слишком часто тестированию в литературе по программной инженерии придается огромное значение, что, собственно, и отражено в определении, данном в Wikipedia: «Тестирование программного обеспечения — это процесс оценки качества компьютерных программ. Тестирование — это практическое техническое изучение, проводимое для того, чтобы предоставить заинтересованным сторонам информацию о качестве продукта или сервиса с учетом контекста, в котором, как предполагается, он будет работать». На самом деле, тестирование программы мало что говорит о ее качестве, поскольку 10 или даже 10 млн тестов — это лишь капля в океане всех возможных случаев.

Безусловно, связь между тестами и качеством программ существует, но она довольно слаба— успешный тест позволяет оценить качество только в том случае, если прежде он не был пройден. Тогда это свидетельствует об отсутствии неудачи и, как правило, — отсутствии самой ошибки\*.

Принцип 1. Определение. Для того чтобы протестировать программу, нужно попытаться заставить ее работать неверно.

В силу этого принципа процесс тестирования обретает цель: его единственная задача — найти ошибки, инициируя неудачное выполнение. Любое умозаключение по поводу качества относится к области гарантии качества, но никак не к области тестирования. Это определение также напоминает нам, что тестирование, в отличие от отладки, не связано с исправлением ошибок, — оно связано только лишь с их поиском.

Тестирование и спецификации

При создании программ, опирающемся на тестирование (test-driven) и получившем известность благодаря методам «скорой» (agile) разработки, тесты ставят на первое место, но иногда считают, что они могут заменить спецификации. Не могут. Тесты, даже если их миллион, — это лишь примеры; в них отсутствует абстракция, которую может дать только спецификация.

Принцип 2. Тесты или спецификации. Тесты не заменяют спецификации.

Опасность заблуждения, что тесты могут выступать в роли спецификаций, была продемонстрирована целым рядом программных катастроф, которые произошли только потому, что никто не подумал об исключительном случае. Несмотря на то, что в спецификациях тоже могут быть упущены из виду какие-то случаи, по крайней мере, в них предпринимается попытка некого обобщения. В частности, спецификации могут использоваться для генерации тестов, даже автоматически (как в случае тестирования, опирающегося на модели); но обратное без вмешательства человека невозможно.

Регрессивное тестирование

Особенностью тестирования, если судить по практике создания программного обеспечения, является печальная склонность к появлению уже исправленных ошибок. Старые головы у гидры, как бы глубоко они ни были отсечены, вырастают снова. Это явление известно как регрессия и заставляет использовать регрессивное тестирование, то есть проверку того, что исправленное ранее по-прежнему работает. Как следствие, вы всегда должны помнить об ошибке, которую уже когда-то обнаружили.

Принцип 3. Регрессивное тестирование. Любое неудачное выполнение должно порождать тестовый случай, который навсегда становится частью тестового пакета данного проекта.

Этот принцип касается всех ошибок, возникших во время разработки и тестирования. Отсюда вытекает необходимость в инструментарии, позволяющем превращать неудачное исполнение в воспроизводимый тестовый случай, и недавно такой инструментарий появился: Contract-Driven Development, ReCrash, JCrasher.

Предсказания

Выполнение теста полезно только в том случае, если вы можете однозначно определить, прошел ли он. Критерий называется предсказанием теста. Если у вас есть несколько десятков или даже сотен тестов, то вы в состоянии проверить их результаты по отдельности, однако с увеличением их количества это становится все менее вероятным. Данная задача требует автоматизации.

Принцип 4. Использование предсказаний. Определение успеха или неудачи тестов должно происходить автоматически.

Эта формулировка оставляет открытым вопрос о форме таких предсказаний. Зачастую предсказания специфицируются отдельно. В исследованиях, таких как наше, они встроенные, поскольку анализируемое программное обеспечение уже включает в себя контракты, согласно которым тесты используются как предсказания.

Принцип 4 (вариант). Контракты как предсказания. Предсказания должны быть частью текста программы как контракты. Успех или неудача теста должны определяться автоматически, причем в рамках этого процесса необходимо вести мониторинг выполнения контракта во время работы программы.

Этот принцип детализирует предыдущий, в силу чего те, кто не использует контракты, могут придерживаться более простой его формы.

Тестовые случаи, проверяемые вручную и автоматически

Многие тестовые случаи проверяются вручную. Тестеры продумывают интересные сценарии выполнения и готовят соответствующие тесты. К этой категории можно добавить тесты, полученные в соответствии с принципом 3 как результат некорректного выполнения, их первоначально не предполагалось использовать при тестировании. Сейчас все чаще можно дополнить эти две категории автоматическими тестовыми случаями, полученными из спецификаций с помощью генератора автоматических тестов. Процесс, ограниченный тестами, выполняемыми вручную, не в полной мере использует возможности современных компьютеров.

Все подходы лишь дополняют друг друга.

Принцип 5. Тестовые случаи, проверяемые вручную и автоматически. Эффективный процесс тестирования должен включать в себя тестовые случаи, проверяемые как вручную, так и автоматически.

Достоинством тестов, выполняемых вручную, является их глубина: они отражают понимание разработчиком имеющегося круга проблем и структуры данных. Преимущество автоматических тестов в их широте: они выполняют проверку большого диапазона значений, в том числе экстремальных, которые люди могут пропустить.

Стратегии тестирования

Теперь мы переходим от практики тестирования к исследованиям новых технологий тестирования, которые, однако, уязвимы в смысле рисков мыслительного процесса. Вы придумали идею, которая, как вам кажется, обещает усовершенствования и подтверждается вашей интуицией. Тестирование — это сложный процесс, и после объективного анализа далеко не все разумные идеи оказываются полезными. Типичный тому пример — случайное тестирование. Интуитивно кажется, что любая стратегия, использующая знания о программе, должна быть лучше случайного ввода. Однако объективные показатели, такие как число найденных ошибок, свидетельствуют о том, что случайное тестирование часто превосходит казавшиеся разумными идеи. В обзоре Ричарда Хемлета, посвященном случайному тестированию (Encyclopedia of Software Engineering, J.J. Marciniak, ed., Wiley, 1994), приводится захватывающее противопоставление человеческого знания и научного анализа.

Ничто не заменяет эмпирические оценки.

Принцип 6. Эмпирические оценки стратегий тестирования. Оценивайте любую стратегию тестирования, однако, какой бы интересной она ни казалась, прибегайте к объективной оценке, используя точные критерии в воспроизводимом процессе тестирования

Принцип 7. Критерий оценки. Самое важное свойство стратегии тестирования — это число обнаруженных ошибок как функция времени.

Функция обнаружения, то есть число ошибок в зависимости от времени fc (t), полезна по двум причинам — используя программную базу с известными ошибками, можно оценить стратегию, посмотрев, сколько ошибок база позволит обнаружить за данное время. Менеджеры проектов могут добавить fc (t) в модель надежности для

## 1.2 Методы тестирования

* Тестирование системы
* Приемочное тестирование
* Нефункциональное тестирование выполняется для проверки производительности, удобства использования, надежности, совместимости приложения. Это включает:
* Тестирование производительности
* Тестирование безопасности
* Юзабилити-тестирование
* Тестирование на совместимость

Ключом к разработке и выпуску высококачественного приложения, которое легко воспринимается конечными пользователями, является создание великолепной среды тестирования, которая может выполнять как функциональное, так и нефункциональное тестирование.

1. Модульное тестирование

Единица — это самая маленькая часть приложения, которую можно протестировать. Целью модульного тестирования является проверка каждого модуля, чтобы увидеть, был ли он разработан в соответствии с требованиями. Единицей может быть отдельная программа, функция, метод и т. Д.

Модульное тестирование часто выполняется разработчиками. Разработчики выполняют модульное тестирование, выполняя кусок кода перед тем, как передать его группе тестирования.

Модульное тестирование является очень важной частью процесса тестирования, потому что, если модульное тестирование выполнено идеально, дефекты выявляются на ранней стадии, и время для устранения этих ошибок сокращается.

2. Интеграционное тестирование

Интеграционное тестирование — это тип тестирования, при котором отдельные блоки группируются и тестируются. Этот тип тестирования проводится для выявления любых видов дефектов во взаимодействии между интегрированными подразделениями или группами.

Все модули объединяются вместе после этапа модульного тестирования для создания компонентов, которые могут выполнять некоторые конкретные задачи. Затем на этих созданных компонентах выполняется интеграционное тестирование, чтобы проверить, ведут ли компоненты себя так, как ожидается, и взаимодействия между блоками бесперебойны. Интеграционное тестирование может выполняться как разработчиками, так и тестерами, либо вручную, либо с использованием автоматизации.

3. Системное тестирование

Системное тестирование выполняется, когда все модули разработаны и интегрированы, чтобы сформировать целостную систему, выполняющую задачу. Системное тестирование проверяет, что система соответствует ее требованиям и работает как ожидалось. Эта полностью интегрированная система может быть определенным интерфейсом или экраном, подобным окну входа в систему.

Этап тестирования системы выполняется группой тестирования, которая тестирует сквозную функциональность приложения, прежде чем приложение кода будет готово к работе.

4. Приемочные испытания

Этап приемочного тестирования — это последний этап процесса функционального тестирования, когда приложение проверяется на готовность к доставке клиенту. Этот этап гарантирует, что приложение соответствует всем бизнес-требованиям и критериям и готово к доставке. Приложение тестируется как тестировщиками из компании, так и за ее пределами (бета-тестеры).

Этап бета-тестирования является наиболее важным этапом, когда компания проверяет продукт на реальных конечных пользователях и устраняет любые обнаруженные дефекты.

5. Тестирование производительности

Тестирование производительности — это метод тестирования, используемый для проверки того, будет ли приложение вести себя должным образом при большой нагрузке и при других условиях. Тестирование производительности проводится для проверки скорости, масштабируемости и стабильности приложения. Тестирование производительности является основным из 4 типов:

* Нагрузочное тестирование
* Стресс-тестирование
* Тест на выносливость
* Спайк тестирование

6. Тестирование безопасности

Тестирование безопасности - это тип тестирования, выполняемый для гарантии того, что данные в приложении безопасны и хорошо защищены. Команда тестирования намеренно пытается найти пути или лазейки, чтобы проникнуть в систему, чтобы получить несанкционированный доступ, который может привести к потере информации или угрозе безопасности. Некоторые из методов тестирования для проверки безопасности:

* целостность
* конфиденциальность
* Аутентификация
* авторизация
* Доступность
* Неотрекаемость

7. Юзабилити-тестирование

Юзабилити-тестирование — это методика тестирования, позволяющая конечным пользователям легко использовать приложение. Юзабилити-тестирование проводится с реальными пользователями, чтобы увидеть, насколько легко им выполнить задачу в приложении.

8. Тестирование на совместимость

Тестирование совместимости проводится, чтобы увидеть, как приложение будет работать в разных средах, таких как различное оборудование, операционные системы, мобильные устройства, платформы, браузеры или условия сети. Цель этого тестирования - убедиться, что приложение не зависит от платформы и выполняет ожидаемые функции в различных средах оценки того, сколько еще ошибок осталось, и ответить на старый вопрос «Когда заканчивать тестирование?»

# 2 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ (Учёт оплаты)

## 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 приложения Учёт оплаты.

Целью тестирования приложения Учёт оплаты 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 работ по тестированию.
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, делать заказы, и просмотра информации.

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 Компьютерный магазин. 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

**Просмотр Отчётов**

Просмотр Списка учащихся

Просмотр Отчётов

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

После установки базы данных можно приступить к ее использованию в тестах. В большинстве простых случаев в наборе тестов есть отдельная база данных, которая совместно используется несколькими тестами в нескольких классах тестирования, поэтому нам нужна некоторая логика, чтобы убедиться, что база данных создана и заполняется ровно один раз в течение всего времени существования тестового запуска.

Существует одна окончательная категория тестов, которая представляет собой дополнительную трудность: тесты, которые изменяют данные, а также явно управляют транзакциями. Поскольку базы данных обычно не поддерживают вложенные транзакции, невозможно использовать транзакции для изоляции, как описано выше, так как они должны использоваться фактическим кодом продукта. Хотя эти тесты, как правило, являются более редкими, необходимо обрабатывать их особым образом: необходимо очистить базу данных до исходного состояния после каждого теста, и параллелизация должна быть отключена, чтобы эти тесты не влияли друг на друга.

Рассмотрим следующий метод контроллера в качестве примера:

[HttpPost]

public ActionResult UpdateBlogUrl(string name, string url)

{

// Note: it isn't usually necessary to start a transaction for updating. This is done here for illustration purposes only.

using var transaction = \_context.Database.BeginTransaction(IsolationLevel.Serializable);

var blog = \_context.Blogs.FirstOrDefault(b => b.Name == name);

if (blog is null)

{

return NotFound();

}

blog.Url = url;

\_context.SaveChanges();

transaction.Commit();

return Ok();

}

Предположим, что по какой-то причине методу требуется сериализуемая транзакция (обычно это не так). В результате мы не можем использовать транзакцию для гарантии изоляции теста. Так как тест фактически зафиксирует изменения в базе данных, мы определим другое приспособление с отдельной базой данных, чтобы убедиться, что мы не вмешиваемся в другие тесты, уже показанные выше:

public class TransactionalTestDatabaseFixture

{

private const string ConnectionString = @"Server=(localdb)\mssqllocaldb;Database=EFTransactionalTestSample;Trusted\_Connection=True";

public BloggingContext CreateContext()

=> new BloggingContext(

new DbContextOptionsBuilder<BloggingContext>()

.UseSqlServer(ConnectionString)

.Options);

public TransactionalTestDatabaseFixture()

{

using var context = CreateContext();

context.Database.EnsureDeleted();

context.Database.EnsureCreated();

Cleanup();

}

public void Cleanup()

{

using var context = CreateContext();

context.Blogs.RemoveRange(context.Blogs);

context.AddRange(

new Blog { Name = "Blog1", Url = "http://blog1.com" },

new Blog { Name = "Blog2", Url = "http://blog2.com" });

context.SaveChanges();

}

}

Это приспособление аналогично описанному выше, но в частности содержит Cleanup метод. Она будет вызывать это после каждого теста, чтобы убедиться, что база данных сбрасывается в начальное состояние.

Если это приспособление будет использоваться только одним классом теста, можно ссылаться на него как на приспособление класса, как описано выше. Однако если поделиться этим приспособлением между несколькими классами, необходимо убедиться, что эти классы не выполняются параллельно, чтобы избежать каких-либо помех. Для этого будет использоваться это как [приспособление коллекции](https://xunit.net/docs/shared-context#collection-fixture) xUnit, а не в качестве [приспособления класса](https://xunit.net/docs/shared-context#class-fixture).

Во-первых, определяем *коллекцию тестов*, которая ссылается на наше приспособление и будет использоваться всеми классами транзакционных тестов, которые требуют его:

[CollectionDefinition("TransactionalTests")]

public class TransactionalTestsCollection : ICollectionFixture<TransactionalTestDatabaseFixture>

{

}

Теперь ссылаемся на коллекцию тестов в нашем тестовом классе и принимаем приспособление в конструкторе, как и раньше:

[Collection("TransactionalTests")]

public class TransactionalBloggingControllerTest : IDisposable

{

public TransactionalBloggingControllerTest(TransactionalTestDatabaseFixture fixture)

=> Fixture = fixture;

public TransactionalTestDatabaseFixture Fixture { get; }

делаем тестовый класс одноразовым, упорядочение метода светильника Cleanup для вызова после каждого теста:

public void Dispose()

=> Fixture.Cleanup();

Как как xUnit только когда-либо создает экземпляр средства сбора один раз, нам не нужно использовать блокировку создания базы данных и заполнения, как сделал выше.

|  |
| --- |
| using System; |
|  | using System.Linq; |
|  | using EF.Testing.BloggingWebApi.Controllers; |
|  | using Xunit; |
|  |  |
|  | namespace EF.Testing.IntegrationTests; |
|  |  |
|  | #region UsingTheFixture |
|  | [Collection("TransactionalTests")] |
|  | public class TransactionalBloggingControllerTest : IDisposable |
|  | { |
|  | public TransactionalBloggingControllerTest(TransactionalTestDatabaseFixture fixture) |
|  | => Fixture = fixture; |
|  |  |
|  | public TransactionalTestDatabaseFixture Fixture { get; } |
|  | #endregion |
|  |  |
|  | #region UpdateBlogUrl |
|  | [Fact] |
|  | public void UpdateBlogUrl() |
|  | { |
|  | using (var context = Fixture.CreateContext()) |
|  | { |
|  | var controller = new BloggingController(context); |
|  | controller.UpdateBlogUrl("Blog2", "http://blog2\_updated.com"); |
|  | } |
|  |  |
|  | using (var context = Fixture.CreateContext()) |
|  | { |
|  | var blog = context.Blogs.Single(b => b.Name == "Blog2"); |
|  | Assert.Equal("http://blog2\_updated.com", blog.Url); |
|  | } |
|  | } |
|  | #endregion |
|  |  |
|  | #region Dispose |
|  | public void Dispose() |
|  | => Fixture.Cleanup(); |
|  | #endregion |
|  | } |

Ниже приведен метод контроллера, который добавляет блог в нашу базу данных:

[HttpPost]

public ActionResult AddBlog(string name, string url)

{

\_context.Blogs.Add(new Blog { Name = name, Url = url });

\_context.SaveChanges();

return Ok();

}

Мы можем протестировать этот метод следующим образом:

[Fact]

public void AddBlog()

{

using var context = Fixture.CreateContext();

context.Database.BeginTransaction();

var controller = new BloggingController(context);

controller.AddBlog("Blog3", "http://blog3.com");

context.ChangeTracker.Clear();

var blog = context.Blogs.Single(b => b.Name == "Blog3");

Assert.Equal("http://blog3.com", blog.Url);

}

Некоторые примечания к приведенному выше коду теста:

Мы запускаем транзакцию, чтобы убедиться, что приведенные ниже изменения не зафиксированы в базе данных и не вмешиваются в другие тесты. Так как транзакция никогда не фиксируется, она неявно откатывается в конце теста при удалении экземпляра контекста.

После внесения нужных обновлений мы очистим средство отслеживания [ChangeTracker.Clear](https://docs.microsoft.com/ru-RU/dotnet/api/microsoft.entityframeworkcore.changetracking.changetracker.clear) изменений экземпляра контекста, чтобы убедиться, что мы фактически загружаем блог из базы данных ниже. Вместо этого можно использовать два экземпляра контекста, но затем необходимо убедиться, что одна и та же транзакция используется обоими экземплярами.

Возможно, вы даже захотите запустить транзакцию в приспособлении CreateContext, чтобы тесты получали экземпляр контекста, который уже находится в транзакции, и готов к обновлению. Это может помочь предотвратить случаи, когда транзакция случайно забыта, что приводит к проверке помех, которые могут быть трудно отлаживать. Кроме того, может потребоваться разделить тесты только для чтения и написать их в различных классах тестирования.

# ЗАКЛЮЧЕНИЕ

Сегодня тестирование является неотъемлемой частью процесса производства программных продуктов. Качественное тестирование помогает своевременно выявлять и исправлять ошибки, тем самым уменьшая риски и затраты на разработку программного обеспечения. При автоматизации тестирования скорость и качество тестирования может повыситься, что приведет к еще большему снижению издержек и повышению качества.

В данной работе была проанализирована эффективность внедрения ИС «Учёт оплаты за обучение» Для достижения поставленной цели были решены такие теоретические задачи, как описание теоретических основ тестирования, классификация и описание различных видов тестирования, описание методологий, анализ процесса тестирования, выявление и описание критериев корректного построения процесса тестирования. В работе также были определены критерии эффективности процесса тестирования, описаны и проанализированы формулы, позволяющие выразить эффективность данного процесса в денежном эквиваленте.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 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.
2. Шнайдер, Роберт Microsoft SQL Server 6.5. Проектирование высокопроизводительных баз данных; М.: Лори, 2010. - 361 c
3. Петкович, Душан Microsoft SQL Server 2012. Руководство для начинающих / Душан Петкович. - М.: БХВ-Петербург, 2012. - 460 c.
4. Тейлор, Аллен SQL для чайников / Аллен Тейлор. - М.: Вильямс, 2014. - 416 c.
5. Браст, Э.Дж. Разработка приложений на основе Microsoft SQL Server 2008 / Э.Дж. Браст. - М.: Русская Редакция, 2010. - 751 c.
6. Хетагуров, Я. А. Проектирование автоматизированных систем обработки информации и управления (АСОИУ). Учебник / Я.А. Хетагуров. - М.: Бином. Лаборатория знаний, 2015. - 240 c.
7. Кристофер, Д. Маннинг Введение в информационный поиск / Кристофер Д. Маннинг, ПрабхакарРагхаван ,ХайнрихШютце. - М.: Вильямс, 2014. - 528 c.
8. Бишоп Дж. С# в кратком изложении; Бином. Лаборатория знаний - М., 2015. - 234 c.
9. ГриффитсИэн Программирование на C# 5.0; Эксмо - М., 2014. - 580 c.
10. Гуриков С. Р. Введение в программирование на языке Visual C#; ИЛ - Москва, 2013. - 448 c.
11. Шилдт Герберт C# 4.0. Полное руководство; Вильямс - М., 2015. - 291 c.
12. Эндрю Троелсен Язык программирования C# 5.0 и платформа .NET 4.5; Диалектика / Вильямс - М., 2015. - 126 c.