Государственное бюджетное профессиональное образовательное учреждение

Самарской области

«Тольяттинский социально-экономический колледж»

**КУРСОВАЯ РАБОТА**

**ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ «УЧЁТ ТОВАРОВ НА СКЛАДЕ ПРИ ПРЕДПРИЯТИИ»**

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

**МДК 05.03 «ТЕСТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ»**

**09.02.07 «ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММИРОВАНИЕ»**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  | **/** | *К.А. Куприянов* |
|  | *подпись* |  | *И.О. Фамилия* |
| \_\_.\_\_.2022 г. |  |  |  |
| Руководитель |  | **/** | *Н.К. Коровина* |
|  | *подпись* |  | *И.О. Фамилия* |
| \_\_.\_\_.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-31

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Куприянова Кирилла Александровича \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия Имя отчество студента*

|  |  |
| --- | --- |
| Тема курсовой работы : | «Учёт товаров на складе при предприятии» |

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

Приложения

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 3 курса ИСП-31 группы Куприянов, К.А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Фамилия, И.О.*

По теме **Учёт товаров на складе при предприятии**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc100908991)

[1 АНАЛИЗ МЕТОДОВ ТЕСТИРОВАНИЯ 6](#_Toc100908992)

[1.1 Критерии и принципы тестирования 6](#_Toc100908993)

[1.2 Методы тестирования 6](#_Toc100908994)

[2 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ (НАЗВАНИЕ) 7](#_Toc100908995)

[2.1. Разработка тестовой документации (тест-дизайн) 7](#_Toc100908996)

[2.2. Разработка тестовых сценариев 7](#_Toc100908997)

[ЗАКЛЮЧЕНИЕ 8](#_Toc100908998)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 9](#_Toc100908999)

[ПРИЛОЖЕНИЯ 10](#_Toc100909000)

# ВВЕДЕНИЕ

Актуальность внедрения автоматизированной информационной системы заключается в инновационном подходе хранения документов в электронной базе данных, является более выгодным вариантом, нежели хранение документов в бумажном формате.

Таким образом, нужно создать программу для учёта находящегося на складе товара. Существует множество СУБД для их редактирования, но хотелось бы создать такое приложение, занимающее небольшой объем, простое в использовании и позволяющее редактировать базу данных без установленной СУБД. Именно этому и посвящена данная курсовая работа.

При изучении предметной области, выяснилось, что информационная система «Учёт товаров на складе при предприятии» 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.

Целью курсовой работы является создание информационной системы «Учёт товаров на складе при предприятии». Для достижения поставленной цели были определены следующие задачи:

* Описать предметную область;
* Спроектировать программный комплекс;
* Выполнить программную реализацию;
* Проанализировать качество программного обеспечения.

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

## 1.1 Критерии и принципы тестирования

Перейдем к основным понятиям

**Тестирование программного обеспечения (Software Testing)** — проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом.  
**Цель тестирования** — проверка соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи программы.  
**Для чего проводится тестирование ПО?**

* Для проверки соответствия требованиям.
* Для обнаружение проблем на более ранних этапах разработки и предотвращение повышения стоимости продукта.
* Обнаружение вариантов использования, которые не были предусмотрены при разработке. А также взгляд на продукт со стороны пользователя.
* Повышение лояльности к компании и продукту, т.к. любой обнаруженный дефект негативно влияет на доверие пользователей.

Принципы тестирования:

* **Принцип 1 — Тестирование демонстрирует наличие дефектов (Testing shows presence of defects)**.  
  Тестирование только снижает вероятность наличия дефектов, которые находятся в программном обеспечении, но не гарантирует их отсутствия.
* **Принцип 2 — Исчерпывающее тестирование невозможно (Exhaustive testing is impossible)**.  
  Полное тестирование с использованием всех входных комбинаций данных, результатов и предусловий физически невыполнимо (исключение — тривиальные случаи).
* **Принцип 3 — Раннее тестирование (Early testing)**.  
  Следует начинать тестирование на ранних стадиях жизненного цикла разработки ПО, чтобы найти дефекты как можно раньше.
* **Принцип 4 — Скопление дефектов (Defects clustering)**.  
  Большая часть дефектов находится в ограниченном количестве модулей.
* **Принцип 5 — Парадокс пестицида (Pesticide paradox)**.  
  Если повторять те же тестовые сценарии снова и снова, в какой-то момент этот набор тестов перестанет выявлять новые дефекты.
* **Принцип 6 — Тестирование зависит от контекста (Testing is context depending)**. Тестирование проводится по-разному в зависимости от контекста. Например, программное обеспечение, в котором критически важна безопасность, тестируется иначе, чем новостной портал.
* **Принцип 7 — Заблуждение об отсутствии ошибок (Absence-of-errors fallacy)**. Отсутствие найденных дефектов при тестировании не всегда означает готовность продукта к релизу. Система должна быть удобна пользователю в использовании и удовлетворять его ожиданиям и потребностям.

## 1.2 Методы тестирования

## Первый метод - тестирование Black-Box

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.

## Типы тестирования черного ящика

Существует много видов тестирования черного ящика, но наиболее важными являются следующие:

* **Функциональное тестирование** — этот тип тестирования черного ящика связан с функциональными требованиями системы; это делают тестеры программного обеспечения.
* **Нефункциональное тестирование.** Этот тип тестирования черного ящика связан не с тестированием конкретной функциональности, а с нефункциональными требованиями, такими как производительность, масштабируемость, удобство использования.
* **Регрессионное тестирование** — [Регрессионное тестирование](https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/ruchnoe-testirovanie/regressionnoe-testirovanie-2) проводится после того, как исправления кода, обновления или любое другое обслуживание системы для проверки того, что новый код не затронул существующий код.

## Второй метод - 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.

## Третий метод - 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.

# 2 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ УЧЁТ ТОВАРОВ НА СКЛАДЕ ПРИ ПРЕДПРИЯТИИ

## 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 учёта контингента учащихся учебного заведения.

Целью тестирования приложения учёта контингента учащихся учебного заведения 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 информации о студентах.

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 учёта контингента учащихся учебного заведения. 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 информации студентов**

Просмотр посещаемости о студентах

Просмотр успеваемости студентах

Выход

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.

При использовании Xunit это можно сделать с помощью [средства](https://xunit.net/docs/shared-context#class-fixture) класса, представляющего базу данных и совместно используемой для нескольких тестовых запусков:

public class TestDatabaseFixture

{

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

private static readonly object \_lock = new();

private static bool \_databaseInitialized;

public TestDatabaseFixture()

{

lock (\_lock)

{

if (!\_databaseInitialized)

{

using (var context = CreateContext())

{

context.Database.EnsureDeleted();

context.Database.EnsureCreated();

context.AddRange(

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

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

context.SaveChanges();

}

\_databaseInitialized = true;

}

}

}

public BloggingContext CreateContext()

=> new BloggingContext(

new DbContextOptionsBuilder<BloggingContext>()

.UseSqlServer(ConnectionString)

.Options);

}

При создании экземпляра приведенного выше средства используется [EnsureDeleted()](https://docs.microsoft.com/ru-RU/dotnet/api/microsoft.entityframeworkcore.storage.idatabasecreator.ensuredeleted#microsoft-entityframeworkcore-storage-idatabasecreator-ensuredeleted) для удаления базы данных (если она существует из предыдущего запуска), а затем [EnsureCreated()](https://docs.microsoft.com/ru-RU/dotnet/api/microsoft.entityframeworkcore.storage.idatabasecreator.ensurecreated#microsoft-entityframeworkcore-storage-idatabasecreator-ensurecreated) для ее создания с помощью последней конфигурации модели ([см. документацию по этим API](https://docs.microsoft.com/ru-ru/ef/core/managing-schemas/ensure-created)). После создания базы данных средство заменит его некоторыми данными, которые могут использовать наши тесты. Стоит потратить некоторое время на размышления о начальных данных, так как изменение его позже для нового теста может привести к сбою существующих тестов.

Чтобы использовать приспособление в тестовом классе, просто реализуйте IClassFixture его над типом светильника, а xUnit внедряет его в конструктор:

public class BloggingControllerTest : IClassFixture<TestDatabaseFixture>

{

public BloggingControllerTest(TestDatabaseFixture fixture)

=> Fixture = fixture;

public TestDatabaseFixture Fixture { get; }

Теперь класс тестирования имеет Fixture свойство, которое можно использовать тестами для создания полнофункциональный экземпляр контекста:

[Fact]

public void GetBlog()

{

using var context = Fixture.CreateContext();

var controller = new BloggingController(context);

var blog = controller.GetBlog("Blog2").Value;

Assert.Equal("http://blog2.com", blog.Url);

}

Вот здесь можно заметить некоторые блокировки в логике создания светильника выше. Если приспособление используется только в одном тестовом классе, оно гарантированно будет создано ровно один раз с помощью xUnit; Но обычно один и тот же компонент базы данных используется в нескольких тестовых классах. xUnit предоставляет [средства сбора](https://xunit.net/docs/shared-context#collection-fixture), но этот механизм предотвращает параллельное выполнение тестовых классов, что важно для производительности тестирования. Чтобы безопасно управлять этим с помощью средства xUnit класса, берем простую блокировку при создании и заполнении базы данных и используем статический флаг, чтобы мы никогда не делали это дважды.

В приведенном выше примере показан тест только для чтения, который является простым случаем с точки зрения изоляции теста: так как ничего не изменяется, вмешательство теста невозможно. В отличие от этого, тесты, которые изменяют данные, являются более проблематичными, так как они могут вмешиваться друг в друга. Одним из распространенных способов изоляции тестов является перенос теста в транзакцию и откат этой транзакции в конце теста. Так как в базе данных ничего не зафиксировано, другие тесты не видят никаких изменений и интерференции не будут избегать.

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");

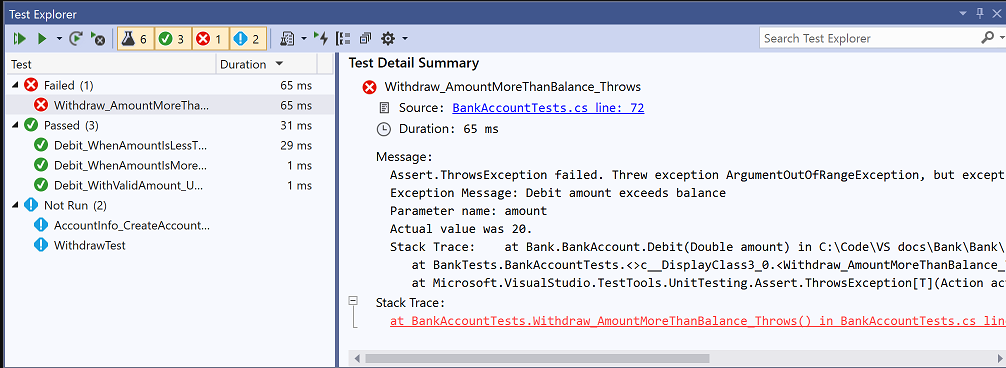
Assert.Equal("http://blog3.com", blog.Url);

}

Некоторые примечания к приведенному выше коду теста:

* 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 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, 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.

При построении проекта тестирования тесты появляются в обозревателе тестов. Если обозреватель тестов не виден, выберите Тест в меню Visual Studio, Windows, затем обозреватель тестов (или нажмите клавиши CTRL + E, T).



Обозреватель модульных тестов

При выполнении, написании и повторном запуске тестов обозреватель тестов может отображать результаты в группах Неудачные тесты, Пройденные тесты, Пропущенные тесты и Незапущенные тесты. Можно выбирать различные группы по параметрам на панели инструментов.

Кроме того, можно фильтровать тесты по совпадению текста в поле поиска на глобальном уровне или с помощью одного из предустановленных фильтров. Можно запустить любую выборку тестов в любое время. Результаты запущенного теста появляются сразу же в строке "успешно/не успешно" наверху окна обозревателя. Детальная информация результата метода тестирования отображается при выборе теста.

# ЗАКЛЮЧЕНИЕ

Обращаем Ваше внимание, что по окончанию исследования подводятся итоги по теме. Заключение носит форму синтеза полученных в работе результатов.

Его основное назначение - резюмировать содержание работы, подвести итоги проведенного исследования. В заключении излагаются полученные выводы, определяется их соотношение с целью исследования, конкретными задачами сформулированными во введении.

Для получения достигнутого результата были реализованы следующие задачи:

* проведен сравнительный анализ аналогов системы;
* проанализирована предметная область, выделены сущности, их атрибуты и связи между ними;
* разработана логическая и физическая модель данных;
* определен функционал системы, выделено два типа пользователя: «Администратор» и «Пользователь».
* создан программный интерфейс в среде программирования MS Visual Studio 2017.

Созданное программное обеспечение позволяет решать основные задачи: внесение новых данных и удаление записей, изменение существующей информации, а также поиск и отбор необходимой информации. Для обеспечения быстроты и удобства предусмотрен анализ данных, в целях просмотра, либо сбора статистики.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Крёнке, Д. Теория и практика построения баз данных. 8-е изд./ Д. Крёнке. — СПб.: Питер, 2013. — 800 с.: ил.
2. Шумаков П.В. ADO.NET и создание приложений баз данных в среде Microsoft Visual Studio.NET/ П.В. Шумаков. –: М.: ДИАЛОГ-МИФИ, 2013. – 258 с.
3. Уильямс, Э. Параллельное программирование на С#/ Э. Уильямс. – СПб.: Москва, 2014. – 672 с.
4. Либерти Д. Программирование на C#/ Д. Либерти. – М.: Издательский дом «Символ-Плюс», 2014. – 688 с.
5. Рихтер Д. Программирование на платформе Microsoft .NET Framework/ Д. Рихтер. – СПб.: Русская редакция, 2013. – 512 с.
6. Дейт, К. Дж. SQL и реляционная теория/ К. Дж. Дейт. – М.: Издательский дом «Вильямс», 2015. – 474 с.
7. Visual Studio 2017 [Электронный ресурс]. – 2003-2017. - Электрон дан. -– Режим доступа: https://www.visualstudio.com/ru-ru/visual-studio-homepage-vs.aspx (дата обращения: 10.04.2018)
8. Вирт, Никлаус. Алгоритмы и структуры данных/ Н. Вирт. – СПб.: БХВ-Петербург, 2015. – 274 с.
9. Кватрани Т. RationalRose 2000 и UML. Визуальное моделирование/ Т. Кватрани. – М.: Издательский дом «Вильямс», 2013. – 583 с.
10. Кумар, В. NET Сетевое программирование/ В. Кумар, Э. Кровчик, Н. Лагари. – М.: Лори, 2014. – 500 с.: ил.
11. Рихтер, Дж. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#/ Дж. Рихтер. – 3-е издание. – СПб.: Питер, 2014. – 400 с.: ил.
12. Микелсен К. Язык программирования C#. Лекции и упражнения/ К. Микелсен. – М.: Издательский дом «ДиаСофт», 2015. – 656 с.
13. Петзольд Ч. Программирование для Microsoft Windows на С#. В 2 томах/ Ч. Петзольд. – СПб.: Русская редакция, 2013. – 536 с.
14. Троелсен Э. C# и платформа .NET. Библиотека программиста/ Э. Троелсен – СПб.: Русская редакция, 2014. – 796 с.
15. Тестирование программного обеспечения [Электронный ресурс] / Информация о тестировании программного обеспечения. – 2003-2017. - Электрон дан. – Режим доступа: http://www.it-courses.by/sqa/ (дата обращения: 10.04.2018).
16. Тейер, Т. Надежность программного обеспечения / Т. Тейер, М. Липов, Э. Нельсон. – М.: Мир, 2015. – 323 с.
17. Тестирование модуля [Электронный ресурс] / Информация о тестировании модуля. – 2003-2017. - Электрон дан. – Режим доступа: http://mgumoscow.blogspot.ru/2012/10/blog-post\_2400.html. (дата обращения: 10.04.2018)