Государственное бюджетное профессиональное образовательное учреждение

Самарской области

«Тольяттинский социально-экономический колледж»

**КУРСОВАЯ РАБОТА**

**ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ**

**«УЧЁТ УЧАСТНИКОВ ОЛИМПИАДЫ»**

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

**МДК 05.03 «ТЕСТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ»**

**09.02.07 «ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММИРОВАНИЕ»**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  | **/** | *Ю.А. Москалёв* |
|  | *подпись* |  | *И.О. Фамилия* |
| \_\_.\_\_.2022 г. |  |  |  |
| 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, выполняемых в рамках МДК.05.03 Тестирование информационных систем**

студента группы ИСП-31

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Москалёва Юрия Александровича\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Фамилия Имя отчество студента*

|  |
| --- |
| Тема курсовой работы: «Учёт участников олимпиады» |

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 Москалёва, Ю.А.

*Фамилия, И.О.*

По теме \_\_\_\_\_\_\_\_\_«Учёт участников олимпиады»\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  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 |  | **/** | *Ю.А. Москалёв* |
|  | *подпись* |  | *И.О. Фамилия* |
| \_\_.\_\_.2022 г. |  |  |  |
| Руководитель |  | **/** | *Н.К. Коровина* |
|  | *подпись* |  | *И.О. Фамилия* |
| \_\_.\_\_.2022 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

Тестирование - очень важный и трудоемкий этап процесса разработки программного обеспечения, так как он позволяет выявить подавляющее большинство ошибок, допущенных при составлении программ. Процесс разработки программного обеспечения предполагает три стадии тестирования:

* автономное тестирование компонентов программного обеспечения;
* комплексное тестирование разрабатываемого программного обеспечения;
* системное или оценочное тестирование на соответствие основным критериям качества.

Формирование набора тестов имеет большое значение, поскольку тестирование является одним из наиболее трудоемких этапов (от 30 до 60 % общей трудоемкости) создания программного продукта. Существуют два принципиально разных подхода к формированию тестовых наборов – структурный и функциональный.

Структурный подход базируется на том, что известны алгоритмы работы программы. В основе структурного тестирования лежит концепция максимально полного тестирования всех маршрутов программы.

Функциональный подход основывается на том, что алгоритм работы программного обеспечения не известен.

Тесты строят, опираясь на функциональные спецификации. Программа рассматривается как «черный ящик», и целью тестирования является выяснение обстоятельств, в которых поведение программы не соответствует требованиям.

Опытные отладчики обнаруживают ошибки путём сравнения шаблонов тестовых выходных данных с выходными данными тестируемых систем. Чтобы определить местоположение ошибки, необходимы знания о типах ошибок, шаблонах выходных данных, языке и процессе программирования. Очень важны знания о процессе разработке программного обеспечения.

# 1 АНАЛИЗ МЕТОДОВ ТЕСТИРОВАНИЯ

## 1.1 Критерии и принципы тестирования

Структурные критерии - используют модель программы в виде "белого ящика", что предполагает знание исходного текста программы или спецификации программы в виде потокового графа управления. Данный класс критериев часто используется на этапах модульного и интеграционного тестирования (Unit testing, Integration testing).

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, он, как правило, 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, поскольку множество ветвей в тестируемом приложении конечно и не так уж велико. 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 1 раз. Если программа содержит цикл (в особенности с неявно заданным числом итераций), то число итераций ограничивается константой (часто - 2, или числом классов выходных путей).

Принципы тестирования:

Принцип 1 — Тестирование демонстрирует наличие дефектов (Testing shows presence of defects). Тестирование может показать, что дефекты присутствуют, но не может доказать, что их нет. Тестирование снижает вероятность наличия дефектов, находящихся в программном обеспечении, но не гарантирует их отсутствия.

Принцип 2 — Исчерпывающее тестирование недостижимо (Exhaustive testing is impossible). 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.

Принцип 3 — Раннее тестирование (Early testing). Чтобы найти дефекты как можно раньше, активности по тестированию должны быть начаты как можно раньше в жизненном цикле разработки ПО или системы, и должны быть сфокусированы на определенных целях.

Принцип 4 — Скопление дефектов (Defects clustering). Разные модули системы могут содержать разное количество дефектов – то есть плотность скопления дефектов в разных частях кода может отличаться. Усилия по тестированию должны распределяться пропорционально фактической плотности дефектов. В основном, большую часть критических дефектов находят в ограниченном количестве модулей.

Принцип 5 — Парадокс пестицида (Pesticide paradox). 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.

Принцип 6 — Тестирование зависит от контекста (Testing is concept depending). Тестирование выполняется 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

В зависимости от доступа разработчика тестов к исходному коду тестируемой программы различают тестирование «белого ящика», «чёрного ящика» и «серого ящика».

При тестировании белого ящика (также говорят — прозрачного ящика), разработчик теста имеет доступ к исходному коду программ и может писать код, который связан с библиотеками тестируемого программного обеспечения. Это типично для компонентного тестирования, при котором тестируются только отдельные части системы. Оно обеспечивает то, что компоненты конструкции работоспособны и устойчивы, до определённой степени. При тестировании белого ящика используются метрики покрытия кода или мутационное тестирование.

При тестировании чёрного ящика тестировщик имеет доступ к программе только через те же интерфейсы, что и заказчик или пользователь, либо через внешние интерфейсы, позволяющие другому компьютеру либо другому процессу подключиться к системе для тестирования. Например, тестирующий компонент может виртуально нажимать клавиши или кнопки мыши в тестируемой программе с помощью механизма взаимодействия процессов, с уверенностью в том, все ли идёт правильно, что эти события вызывают тот же отклик, что и реальные нажатия клавиш и кнопок мыши. Как правило, тестирование чёрного ящика ведётся с использованием спецификаций или иных документов, описывающих требования к системе. Обычно в данном виде тестирования критерий покрытия складывается из покрытия структуры входных данных, покрытия требований и покрытия модели (в тестировании на основе моделей).

При тестировании серого ящика разработчик теста имеет доступ к исходному коду, но при непосредственном выполнении тестов доступ к коду, как правило, не требуется.

**2 ТЕСТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ «УЧЁТ УЧАСТНИКОВ ОЛИМПИАДЫ»**

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

Тест дизайн — один из первоначальных этапов тестирования программного обеспечения, этап планирования и проектирования тестов. Тест дизайн представляет собой продумывание и написание тестовых случаев (test case), в соответствии с требованиями проекта, критериями качества будущего продукта и финальными целями тестирования.

Целью тестирования ИС «Учёт участников олимпиады» является проверка корректной работы.

Итогом процесса тестирования будут следующие материалы:

* Определить существующую информацию о проекте и программных компонентах, подлежащих тестированию;
* Описать стратегии тестирования, которые будут использоваться;
* Определить необходимые ресурсы для проведения работ по тестированию;
* Привести результаты тестирования.

Результаты будут отправлены заказчику в виде отчетов.

Условия для тестирования. Приложение должно удовлетворять потребность пользователя в активностях, связанных с просмотром информации об олимпиадах и участниках.

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. 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 Критерии 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

2.1 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

* Методом «Чёрного ящика»;
* Проверка выходных данных.

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.

Нагрузочное и тестирование безопасности не будет проведено 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 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). 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, 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");

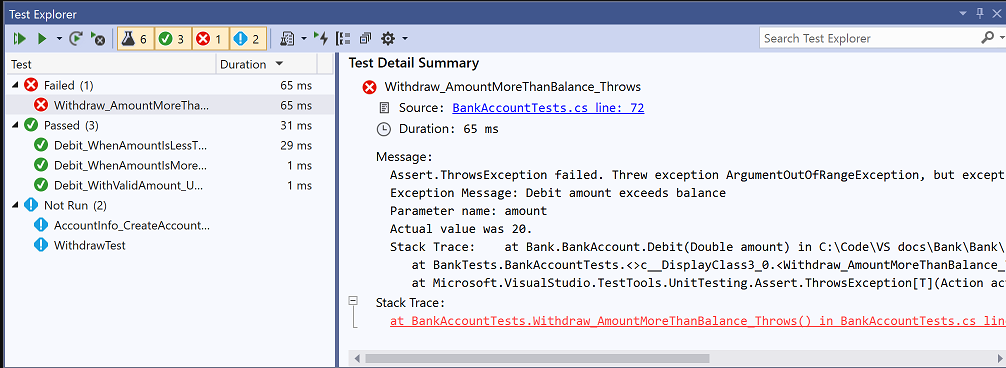
Assert.Equal("http://blog3.com", blog.Url);

}

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**

В результате работы можно сделать вывод, что тестирование является одним из немаловажных инструментов обеспечения качества программного продукта. Именно поэтому для удовлетворения желаний заказчиков, гарантии безопасности пользователей, а также сокращения времени и ресурсов на разработку информационных систем необходимо провести анализ и грамотно построить процесс тестирования.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Джек Фолк, Сэм Канер, Енг. Кек Нгуен. Тестирование программного обеспечения. Издательство ДиаСофт, 2001.
2. Савин Роман. Тестирование DOT COM. Издательство Дело, 2007.
3. Виды Тестирования [Электронный ресурс]/ Про Тестинг - Тестирование Программного Обеспечения. URL: http://www.protesting.ru/testing/types/sanity.html
4. Certifying Software Testers Worldwide [Электронный ресурс]/ URL: http://www.istqb.org/
5. Александр Хрущев. Эффективность использования автоматических тестов в ИТ-проектах. Доклад на конференции CEE-SECR 2009, октябрь 2009.
6. Оценка эффективности автоматизации тестирования [Электронный ресурс]/ Технологии качества. URL: http://a1qa.ru/blog/otsenka-effektivnosti-avtomatizatsii-testirovaniya/
7. Джефф Рэшка, Элфрид Дастин, Джон Пол. Автоматизированное тестирование программного обеспечения. Внедрение, управление, эксплуатация. Издательство Лори, 2012.
8. Ron Patton. Software Testing. 2005.
9. Винниченко И.В. Автоматизация процессов тестирования. Издательство Питер, 2005.
10. Рекс Блек. Ключевые процессы тестирования - М.: Издательство Лори, 2014. - 544 с.
11. Основные положения тестирования [Электронный ресурс]/ Интересные публикации / Хабрахабр. URL: https://habrahabr.ru/post/110307/
12. Что такое Конфигурационное тестирование [Электронный ресурс]/ software-testing. URL: http://software-testing.org/testing/chto-takoe-konfiguracionnoe-testirovanie-configuration-testing.html
13. Автоматизация тестирования [Электронный ресурс]/ Перфоманс Лаб. URL: http://www.performance-lab.ru/avtomatizacija-testirovanija