Министерство образования Республики Беларусь

УО «Полоцкий государственный университет»

Кафедра технологий программирования

**О Т Ч Е Т**

о прохождении практики

студента 4 курса группы 10-ИТ-1

Гонтарева Константина Витальевича

в период с 09.06.2014 по 04.07.2014

Наименование базы практики: ГУО «Лицей г. Новополоцк»

Руководитель от предприятия: заместитель директора

(должность) Тверитнева А.А.

Руководитель от кафедры: старший преподаватель

(ученая степень, звание) Бурачёнок И.Б.

Практика защищена

с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Полоцк, 2014

**Содержание**

Введение 5

1 Общая ознакомительная часть практики 6

1.1 Общие сведения об предприятии ИООО «Эпам Системз» 6

1.2 Организационная структура управления предприятия ИООО «Эпам Системз» 7

2 Основная часть 9

2.1 Постановка задачи проектирования 9

2.2 Описание предметной области 10

2.3 Сравнительный анализ аналогов и прототипов 10

2.4 Выбор и обоснование средств и методов решения задач. 11

2.5 Разработка алгоритмов решения задач 13

2.6 Проектирование программного обеспечения 16

2.7 Реализация и тестирование 20

Заключение 23

Список использованной литературы 24

# Введение

# 1 Общая ознакомительная часть практики

## 1.1 Общие сведения об предприятии

## 1.2 Организационная структура управления предприятия

# 2 Основная часть

## 2.1 Постановка задачи

Необходимо создать программное средство, обеспечивающее автоматизированную проверку олимпиадных задач по программированию в ГУО «Лицей г.Новополоцка». Решенная олимпиадная задача представляет собой исполняемый файл, принимающий входные данные в виде текстового файла input.txt, находящегося в одной аудитории с ним. Выходные данные представляют собой текстовый файл output.txt в той же директории. Ниже перечислены основные требования к разрабатываемому программному обеспечению:

1. Возможность создания набора задач, содержащих различные условия и пары тестовых значений, а также возможности управления этим набором;
2. Возможность потоковой проверки сразу нескольких исполняемых файлов по парам тестовых данных проверяемой задачи;
3. Отображение процесса проверки исполняемых файлов по парам тестовых данных в реальном времени;
4. Обеспечить возможность остановить процесс проверки;
5. Обеспечить возможность сохранения и загрузки набора задач и результатов проверки;
6. Интуитивно понятный пользователю интерфейс;
7. Высокий уровень расширяемости и высокие показатели метрик кода.

В мои обязанности входит разработка независимой от пользовательского интерфейса библиотеки, отвечающей за функциональность разрабатываемого программного средства, а также API для данной библиотеки.

## 2.2 Описание предметной области

Предметная область решаемой задачи включает в себя следующие элементы:

1. Задача для проверки. Характеризуется названием и условием. Является центральным объектом реализуемого программного средства;
2. Пара тестовых данных. Под парой тестовых данных подразумевается набор входных и выходных данных, представленных в виде набора строковых данных. Каждая пара тестовых данных относится к некоторой задаче;
3. Требование к задаче. Оно представляет собой некоторую характеристику и её значение, необходимое для того, чтобы тест был успешно пройден;
4. Решение - это исполняемый файл, который может включать в себя различную информацию, такую как имя ученика, время проверки и др. Но в нашем программном средстве в такой информации нет необходимости, и под решением мы будем подразумевать путь к исполняемому файлу;
5. Результат проверки теста. Этот элемент характеризуется информацией об успешности прохождения теста, а также о выполнении заявленных в задаче требований;
6. Результат проверки задачи. Представляет собой набор результатов проверок тестов, а также информацию о решении и задаче, к которым относится этот результат.

Описанных выше элементов предметной области достаточно для реализации требуемого программного обеспечения и выполнения требований к нему.

## 2.3 Сравнительный анализ аналогов и прототипов

На сегодняшний день существует множество онлайн решений для конкретных языков программирования и конкретных задач, однако наиболее приближенным аналогом реализуемого средства является система проверки задач Ejudge.

Ejudge является web-приложением, устанавливаемая на ОС Linux. Функциональность приложения практически идентична разрабатываемому программному средству, но из-за web направленности имеет некоторые недостатки:

* Нельзя отслеживать процесс тестирования;
* Отсутствует возможность остановить тестирование.

Из плюсов можно отметить наличие в системе учеников и организованного взаимодействия с ними.

Рассмотрев аналоги, было решено реализовывать настольное приложение, т.к. это позволяет наиболее эффективно использовать ресурсы компьютера, а также избавиться от потребности в использовании сети для работы с приложением.

## 2.4 Выбор и обоснование средств и методов решения задач.

Исходя из задачи, для проектирования реализуемого программного средства было решено использовать Enterprise Architect 9.0. Данное case-средство позволяет разрабатывать необходимые UML диаграммы, а также имеет средства прямого и реверсивного инжиниринга.

В качестве используемого в разработке языка программирования был выбран язык C#. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, LINQ, исключения, комментарии в формате XML, а также множество других возможностей.

Также, для разработки пользовательского интерфейса было решено использовать технологию WPF (Windows Presentation Foundation). В основе WPF лежит векторная система визуализации, не зависящая от разрешения устройства вывода и созданная с учётом возможностей современного графического оборудования. WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (Extensible Application Markup Language), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили и множество других возможностей. Графической технологией, лежащей в основе WPF, является DirectX, в отличие от Windows Forms, где используется GDI/GDI+. Производительность WPF выше, чем у Windows Forms за счёт использования аппаратного ускорения графики через DirectX.

Основным средством разработки, было решено использовать IDE Visual Studio 2013 Ultimate. Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Иммется встроенный отладчик уровня исходного кода. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как например, Subversion и Git), добавление новых наборов инструментов. Также в данной редакции этой IDE присутствует возможность просмотреть проект визуально, в виде карты кода, проводить модульное тестирование и анализ исходного кода на предмет ошибок, рассчитывать метрики кода.

В качестве дополнительного средства для разработки пользовательского интерфейса было решено использовать Microsoft Expression Blend 4. Данное ПО от корпорации Microsoft, предназначено для разработки дизайна веб-интерфейсов и графических настольных приложений, совмещает в себе особенности этих двух типов приложений. Интерактивный WYSIWYG-редактор для дизайна интерфейсов, основанных на XAML-приложениях, для Windows Presentation Foundation и приложений Silverlight.

В качестве системы управления версиями, было решено использовать Github. Github - крупный веб-сервис для хостинга IT-проектов и их совместной разработки, который основан на системе контроля версий Git.

## 2.5 Разработка алгоритмов решения задач

Основная задача реализуемого программного средства – проверка исполняемого файла на генерацию правильных выходных данных, поэтому рассмотрим алгоритм проведения этого процесса.

1. Получить данные о решении задачи, т.е. о тестируемом исполняемом файле;
2. Получить пару тестовых значений, полученных ранее от пользователя;
3. Дать пользователю понять, что тест запущен;
4. Загрузить входные данные пары тестовых значений в файл input.txt в ту же директорию, где находится исполняемый файл;
5. Запустить исполняемый файл и ожидать завершения его выполнения;
6. Сравнить выходные данные пары тестовых значений с данными, записанными в файл output.txt в директории исполняемого файла;
7. Сохранить результат в базе результатов;
8. Дать пользователю понять, что тест закончен;

Учитывая необходимость проверки сразу большого набора тестов с разными исполняемыми файлами и парами тестовых значений, разработаем алгоритм, выполняющий такое тестирование:

1. Поместить все исполняемые файлы в очередь исполняемых файлов;
2. Поместить все тестовые пары в очередь тестовых пар;
3. Пока в очереди исполняемых файлов и в очереди тестовых пар есть элементы:
   1. Дать пользователю понять, что начался следующий тест;
   2. Провести тест с текущей тестовой парой и исполняемым файлом;
4. Дать пользователю понять, что все тесты завершены.

В описанных выше алгоритмах также показано, каким образом должен оповещаться пользовательский интерфейс, т.к. напрямую следить за процессом тестирования из него нельзя. Это обусловлено тем, что нельзя выполнять длительные операции в одном потоке с пользовательским интерфейсом, т.к. это приведёт к его блокировке.

Одним из важных задач разрабатываемого программного средства является хранение информации о задачах, тестах и результатах тестов. Для решения этой задачи было решено использовать сериализацию данных по запросу в формат JSON. JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript. Данный формат, по сравнению с другими обеспечивает оптимальный занимаемый данными объем и читаемость. Для .Net существует множество библиотек для работы с этим форматом, однако мы решили использовать Newtonsoft.Json, являющейся одной из самых популярных в своём роде.

Разработанные алгоритмы полностью описывают процесс тестирования и готовы к реализации на используемом языке программирования.

## 2.6 Проектирование программного обеспечения

Начинать проектирование будем с представления системы в целом. Для создания наиболее эффективного программного средства, необходимо максимально отделить логику его работы от пользовательского интерфейса. Было решено разделить реализуемое приложение на 2 части – TTS.Core и TTS.UI. На рис 1 представлена диаграмма развертывания, на которой можно отследить основные зависимости реализуемых и используемых библиотек.



Рис.1 Диаграмма развертывания

Можно выделить следующие модельные сущности, которые будут использоваться в системе:

* Задача. Её атрибуты:
  + 1. Имя;
    2. Описание;
    3. Пары тестовых значений;
    4. Требования;
* Пара тестовых значений. Её атрибуты:
  + 1. Входные данные;
    2. Выходные данные.
* Характеристика. Её атрибуты:
  + 1. Тип;
    2. Значение.
* Результат проверки задачи. Его атрибуты:
  + 1. Связанная задача;
    2. Путь к исполняемому файлу;
    3. Результаты прохождения тестов.
* Результат прохождения теста. Его атрибуты:
  + 1. Связанный тест;
    2. Показатель успешности;
    3. Характеристики;

Также существует ряд служебных сущностей, отвечающих за управление модельными сущностями. Полный набор сущностей представлен на диаграмме классов в проекте Enterprise Architect на прилагаемом диске.

Необходимо четко понимать, что очень важно инкапсулировать всю работу библиотеки TTS.Core по отношению к TTS.UI, для чего в библиотеке будет реализован ряд интерфейсов и класс CoreAccessor, обеспечивающий взаимодействие между TTS.Core и любым другим внешним пользователям за счет предоставления им инстанцированных объектов в виде объекта соответствующего интерфейса. На рисунке 2 представлена диаграмма классов, предоставляемых внешним пользователям.



Рис. 2 – Диаграмма классов

При проектировании было выделено 3 сущности, отвечающих за процесс тестирования: TestController, TestPerformer, IOTest.

TestController ответственен за выполнение тестов в целом, за ведение очередей тестовых пар и исполняемых файлов, за передачу этих данных в объект класса TestPerformer и за формирование результата проверки задачи. Класс TestPerformer отвечает непосредственно за процесс тестирования. Именно TestPerformer формирует результат проверки теста и проводит процесс тестирования. Класс IOTest входит в TestPerformer и отвечает непосредственно за проверку исполняемого файла на соответствие парам тестовых значений. Таким образом в TestPerformer в будущем может быть расширен для вычисления других характеристик исполняемого файла. На рисунке 2 представлена диаграмма последовательности процесса тестирования для этих трёх классов.



Рис.2 Диаграмма последовательности процесса тестирования

При проектировании использовались следующие шаблоны проектирования: стратегия, одиночка, наблюдатель, фабрика, медиатор.

## 2.7 Реализация и тестирование

Листинг . Открытые методы класса TestController

public void Run(IList<Guid> tests, IList<string> files)

{

DataManager storage = DataManager.Instance;

this.testsToPerform = storage.Tests.Where(test => tests.Contains(test.ID)).ToList();

this.SetupFilesQueue(files);

this.ProceedNextFile();

}

public void Stop()

{

this.testInfoQueue.Clear();

this.filesQueue.Clear();

this.cancelationPending = true;

}

Листинг . Конструктор Task

public Task()

{

this.id = Guid.NewGuid();

this.name = String.Empty;

this.description = String.Empty;

this.requirements = new List<Characteristic>();

this.tests = new List<Guid>();

}

Листинг . Запуск метода Start класса IOTest

public void Start(BackgroundWorker worker, DoWorkEventArgs args)

{

try

{

if (!this.IsReady)

throw new InvalidOperationException("The controller is not ready!");

this.PrepareInput(worker);

this.PerformProcess(worker);

this.CheckOutput(worker);

this.DeleteFiles(worker);

}

catch (Exception exc)

{

throw new Exception("The test was interrupted by error!", exc);

}

}

Листинг . Метод сериализации данных SaveStorage класса IDataManager

private void SaveStorage(string path, Storage storage)

{

if (File.Exists(path))

File.Delete(path);

using (TextWriter writer = new StreamWriter(File.Create(path)))

{

JsonSerializer serializer = new JsonSerializer();

serializer.Serialize(writer, storage);

}

}

# Заключение

В результате прохождения практики была изучена информация о предприятии и структура предприятия ИООО «Эпам Системз», действующие нормы по технике безопасности, документация. В результате был создан прототип программного приложения турагентство. Был получен огромный опыт работы в команде, совместного проектирования и разработки программного обеспечения. Были изучены такие технологии как unit, checkstyle, log4j, findbag. Были получены практические навыки в создание программы на языке Java и реализация паттерна MVC, использования SVC в качестве системы управления версиями. Также были углублены знания в таких технологиях как: tomcat, J2EE, PostgreSql.

# Список использованной литературы

1. Шилдт Г. Полный справочник по SQL.: Пер. с англ. – М.: ООО “И.Д. Вильямс”, 2004. – 752 c.: ил.
2. Макконнелл С.. Совершенный код. Мастер класс. - М: Русская редакция – 896с.: ил.
3. Экель Б. – Философия Java. – СПб.: Питер, 2013. – 640 с.: ил.
4. Хорстманн К., Корнелл Г.. – Java2. Библиотека профессионала. Том 1.: Пер. С англ.\Мухин Н.. - Вильямс. - 816 с.: ил.
5. Хорстманн К., Корнелл Г.. – Java2. Библиотека профессионала. Том 2. Тонкости программирования.: Пер. с англ.\Мухин Н.. – Вильямс, 2012. - 912с.: ил.