Содержание

[Содержание 0](#_Toc451274002)

[ВВЕДЕНИЕ 1](#_Toc451274003)

[**1** ОБЗОР ЛИТЕРАТУРЫ 3](#_Toc451274004)

[**1.1** Анализ существующих систем тестирования алгоритмов классификации наборов тестовых данных 3](#_Toc451274005)

[**1.2** Обзор используемых технологий 4](#_Toc451274006)

[**1.3** Выбор системы контроля версий 11](#_Toc451274007)

[**2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 14](#_Toc451274008)

[**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 19](#_Toc451274009)

[**3.1** Разработка базы данных 19](#_Toc451274010)

[**3.2** Разработка уровня доступа к данным 23](#_Toc451274011)

[**3.3** Разработка генератора тестовых наборов данных 25](#_Toc451274012)

[**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 33](#_Toc451274013)

[**4.1** Описание алгоритма хэширования паролей 33](#_Toc451274014)

[**4.2** Описание алгоритма регистрации и авторизации пользователей 38](#_Toc451274015)

[**4.3** Описание алгоритма генерации случайных чисел 39](#_Toc451274016)

[**4.4** Описание алгоритма получения последовательности с нужным распределением 42](#_Toc451274017)

[**7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ WEB-СИСТЕМЫ ДЛЯ ТЕСТИРОВАНИЯ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ 45](#_Toc451274018)

[**7.1** Введение 45](#_Toc451274019)

[**7.2** Расчет объема ПО 45](#_Toc451274020)

[**7.3** Расчет нормативной трудоемкости, заработной платы исполнителей 47](#_Toc451274021)

[7.4 Расчет экономического эффекта от применения программного средства у пользователя 53](#_Toc451274022)

[**7.5** Вывод 58](#_Toc451274023)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 58](#_Toc451274024)

ВВЕДЕНИЕ

Машинное обучение является обширным подразделом искусственного интеллекта. Одной из задач, решаемых с помощью машинного обучения, является классификация различных наборов тестовых данных. В частности, существующих алгоритмов классификации.

Алгоритмом классификации принято называть формализованную задачу, в которой имеется входной набор данных (объектов), которые необходимо разбить на классы по определенным критериям. Также имеется конечное множество объектов, для которых известно, к каким классам они относятся. Это множество считается выборкой.

Задача заключается в построении алгоритма, способного определить классовую принадлежность объекта из входного набора.

Целью задачи классификации, как правило, является частичная или полная автоматизация решения сложных профессиональных задач в различных областях человеческой деятельности.

В современном мире машинное обучение имеет достаточно широкий спектр применения. Оно используется в таких областях, как распознавание речи и рукописи, техническая и медицинская диагностика, распознавание образов, обнаружение мошенничества и спама и так далее.

Количество сфер человеческой деятельности, в которых не только возможно, но и необходимо применение машинного обучения, растет с каждым днем. Непрерывная повсеместная информатизация приводит к накоплению огромных объемов данных в науке, бизнесе, производстве, здравоохранении, транспорте. Вместе с этим возникают такие задачи, как управление, прогнозирование, принятие решений. Они, как правило, сводятся к выполнению алгоритмов машинного обучения.

В настоящее время существует такие системы оценки алгоритмов машинного обучения, как Полигон(Россия), TunedIt(Польша), MLComp (США).

Данные системы имеют ряд ограничений. Например, система TuneIt позволяет пользователю работать только с алгоритмами, описанными на языке Java. MLComp лишена данного недостатка, т.к. в качестве входного параметра принимает скомпилированный исполняемый файл. Однако исполняемый файл должен быть совместимым с операционной системой Linux.

Существует также программное средство TestSystem, которое является собственной разработкой белорусского программиста.TestSystem имеет удобный и интуитивно понятный пользователю web интерфейс. Данная система позволяет реализовывать свои алгоритмы на любом языке. Тестирование происходит на стороне пользователя. Это позволяет сохранить приватность коммерческих алгоритмов.

Целью данного дипломного проекта является доработка и расширение функционала программного средства TestSystem.

Планируется расширение разнообразия алгоритмов тестирования данных.

Также в рамках данного дипломного проекта будет вставлен расчет дополнительных статистических параметров оцениваемых алгоритмов.

В имеющуюся систему будет добавлен модуль автоматической генерации входных тестовых данных.

Система должна оставаться расширяемой, с возможностью добавления новой функциональности, а также содержать тесты на все добавленные модули.

**1** ОБЗОР ЛИТЕРАТУРЫ

**1.1** Анализ существующих систем тестирования алгоритмов классификации наборов тестовых данных

В настоящее время известно несколько систем тестирования алгоритмов классификации [1].

Одной из таких систем является [RapidMiner](http://www.machinelearning.ru/wiki/index.php?title=RapidMiner). Для работы с ней пользователю необходимо установить дополнительное программное обеспечение локально на собственном компьютере. Это оставляет ему возможность по-своему реализовать методику тестирования, модифицировать исходные данные, а в некоторых случаях – и сами алгоритмы. Как следствие, эмпирические результаты, представленные в разных статьях, оказываются несопоставимыми, даже если тестирование проводилось на одних и тех же задачах.

Еще одним известным программным средством оценки методов машинного обучения является «Полигон» (Россия) [2]. Система не накладывает ограничений на язык программирования. Однако программисту необходимо разработать слой взаимодействия собственного алгоритма с сервисом.

Плюсом системы Полигон является также реализация алгоритма кросс-валидации, что обеспечивает воспроизводимость тестов и статистическую надежность результатов[3].

Существует также программное средство TuneIt [4], разработанное в Польше. Оно включает в себя три компонента: базу знаний, репозиторий и приложение, запускающее алгоритмы тестирования.

Приложение отрабатывает на стороне сервера и работает только с исходным кодом, описанным на языке Java, что снижает скорость выполнения алгоритмов и затрудняет перенос в систему готовых алгоритмов, написанных на других языках.

Weka — библиотека алгоритмов [машинного обучения](http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D1%83%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для решения задач [интеллектуального анализа данных](http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D0%BB%D0%BB%D0%B5%D0%BA%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) [5]. Аналогично с TuneIT, программа накладывает ограничение на язык разрабатываемого алгоритма, позволяя использовать исключительно Java.

[RapidMiner](http://www.machinelearning.ru/wiki/index.php?title=RapidMiner), [WEKA](http://www.machinelearning.ru/wiki/index.php?title=WEKA) являются системами с открытым исходным кодом. Следовательно, в них невозможно использовать коммерческие алгоритмы. В то же время, их владельцы вполне могли бы предоставлять их для тестирования, которое не предполагает извлечения коммерческой выгоды. Как раз наоборот, сами владельцы получат выгоду от рекламы, разумеется, если их алгоритм действительно работает хорошо.

Test System – система оценки качества алгоритмов машинного обучения, разработанная белорусским программистом. Она имеет интуитивно понятный и простой веб-интерфейс, не накладывает ограничений на язык разработанного алгоритма. Для запуска пользователю достаточно web-браузера с поддержкой JavaScript и Cookies.

При загрузке тестовых данных, пользователю необходимо придерживаться формата тестов, который описан в Help секции сайта. При оценивании алгоритма пользователь должен настроить свою программу для работы с тестовыми наборами определенного формата.

Интерфейс программного средства Test System представлен на рисунке 1.1.

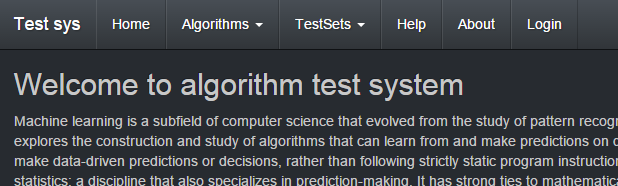


Рис. 1.1 – Интерфейс программного средства Test System

Система является расширяемой. Для дополнительного удобства работы с ней планируется добавление автоматического генератора тестовых данных, а также добавление новых алгоритмов тестирования.

**1.2** Обзор используемых технологий

В наши дни трудно представить себе человека, жизнь которого никогда не затрагивало стремительное развитие веб-технологий. Неважно будь то это сфера развлечений, финансов или, например, образовательная сфера, для разработки и сопровождения современных сервисов везде используются передовые подходы конструирования программных модулей, позволяющих практически с нуля создавать сложнейшие программные системы. Активная эксплуатация широко популярных веб-платформ, таких как ASP .NET, Django, Hibernate, Spring, Ruby on Rails и др. является передовым подходом в области эффективного веб-программирования.

Выбор наиболее подходящей языковой платформы для решения технически грамотно поставленных задач требует обширного опыта в области разработки сложных программных систем, в особенности проектирования гибких, надёжных и интегрируемых программных продуктов. Подобный выбор релевантного программного инструментария, а соответственно и в дальнейшем используемых технологий, является одним из самых нетривиальных и интересных этапов разработки программного обеспечения.



Рисунок 1.2 – Разнообразие актуальных веб-технологий

Существенное преимущество построения Web приложений для поддержки стандартных функций браузера заключается в том, что функции должны выполняться независимо от операционной системы данного клиента. Вместо того чтобы писать различные версии для Microsoft Windows, Mac OS X, GNU/Linux и других операционных систем, приложение создается один раз для произвольно выбранной платформы и на ней разворачивается. Однако различная реализация HTML, CSS, DOM и других спецификаций в браузерах может вызвать проблемы при разработке веб-приложений и последующей поддержке. Кроме того, возможность пользователя настраивать многие параметры браузера (например, размер шрифта, цвета, отключение поддержки сценариев) может препятствовать корректной работе приложения.

Таким образом, выбор наиболее подходящих технологий для эффективной реализации технических задач предстаёт в виде первостепенного вопроса.

Для создания программного модуля оценки эффективности алгоритмов классификации наборов тестовых данных понадобилось использовать Visual Studio 2015 в качестве среды разработки, как наиболее популярный, мощный и сопровождаемый полным пакетом документации продукт разработки программного обеспечения. По словам представителей корпорации Microsoft Visual Studio 2015 это замечательный шаг вперед в области разработки. Совмещая инновации и продуктивность этот инструментарий позволяет создавать решения самого широкого спектра.

Особенностью удобной разработки в среде Visual Studio 2015 является ещё и то, что данная среда объединяет отладку и профилирование в единый диагностический хаб, который позволяет проверить корректность реализации и параметры производительности любого приложения в режиме контекстной отладки. В дополнение к этому в среде реализованы подсказки в виде Perf Tips для того, чтобы под рукой всегда была информация о производительности реализуемых функций. В редакторе кода разработчики имеют возможность использовать всю мощь инструментария, базирующегося на технологии Roslyn, включая Light Bulbs которые вовремя подсказывают удобное решение для участка кода и помогают в его рефакторинге. Благодаря Roslyn и анализаторам кода, существует возможность модифицировать и настраивать предупреждения и подсказки прямо в редакторе. Все эти особенности делают разработку в среде Visual Studio 2015 максимально удобной.

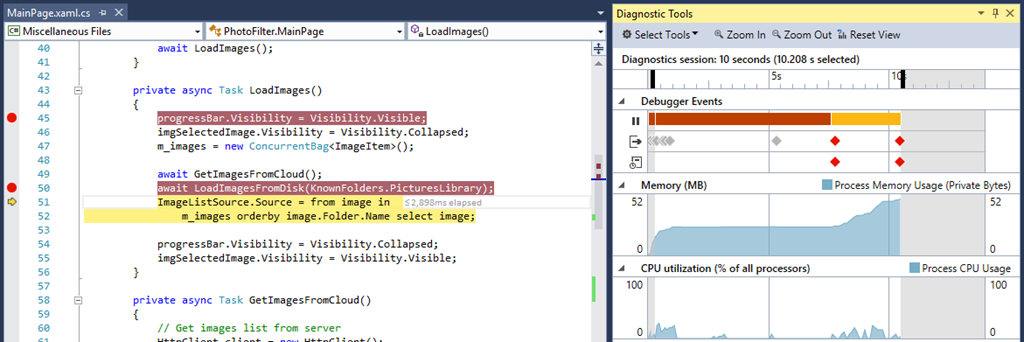


Рисунок 1.3 – Удобное редактирование кода в среде Visual Studio 2015

Ещё одна причина выбора данной среды разработки является её существенно высокая производительность по сравнению с более ранними версиями программы.

Среда разработки Visual Studio 2015 поддерживает работу с кодом на языке С# (C Sharp), который активно эксплуатировался в качестве языкового инструмента разработки программного модуля дипломного проекта, созданного на базе платформы .NET Framework 4.5.

Платформа .NET Framework состоит из нескольких частей: общеязыковой исполняющей среды (common language runtime, CLR) и библиотеки классов Framework Class Library (FCL).

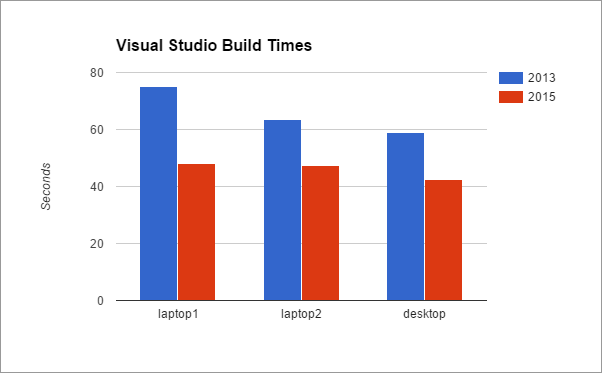


Рисунок 1.4 – Гистограмма производительности Visual Studio 2015

CLR обеспечивает интеграцию языков и позволяет объектам, благодаря стандартному набору типов и метаданным, созданным на одном языке, быть равноправными компонентами кода, написанного на другом языке. Другими словами, CLR этот тот самый механизм, который позволяет программе выполняться в нужном нам порядке, вызывая функции управления данными. CLR управляет процессом выполнения команд машинного кода и решает, какой кусок кода или функцию от куда взять и куда подставить прямо в момент работы программы.

IL (Intermediate Language) — это код на специальном стековом языке, напоминающий по своей сути ассемблер, но написанном для .NET Framework платформы. В данный язык преобразуется код из других языков более верхнего уровня, таких как С#, Visual Basic, Jscript и Fortran. На данном этапе пропадает зависимость от конкретного выбранного языка. Весь код преобразуется в IL в рамках соответствия общей языковой спецификации CLS.

CLS (Common Language Specification) — это языковая спецификация, выпущеная корпорацией Microsoft. Она описывает минимальный набор возможностей, которые должны реализовать производители компиляторов, чтобы их продукты работали в CLR. CLR/CTS поддерживает больше возможностей, определенных CLS. Ассемблер IL поддерживает полный набор функций CLR/CTS. Такие языки как С# и Visual Basic поддерживают часть возможностей CLR/CTS, в том числе минимум от CLS.

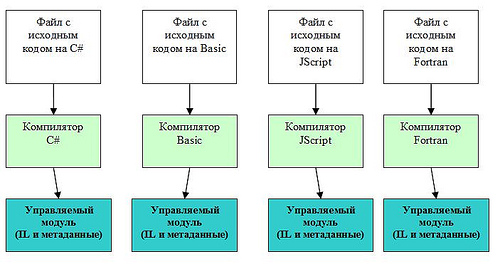


Рисунок 1.5 – Процесс компиляции кода при помощи CLR

CTS (Common Type System) — это общая система типов в CLR, а также это стандарт, признанный ECMA, который описывает определение типов и их поведение. CTS определяет правила наследования, правила виртуальных методов и правила времени жизни объектов. В отличие от С++, CTS поддерживает только единичное наследование. Все типы наследуются от System.Object (Object — имя типа, корень все остальных типов, System — пространство имен).

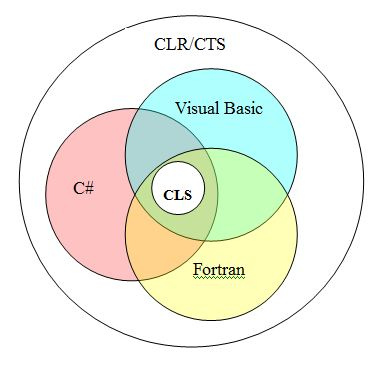


Рисунок 1.6 – Пример проверки на соответствие CLS

Метаданные — это набор из таблиц данных, описывающих то, что определено в модуле. Также существуют таблицы, указывающие на что ссылается управляемый модуль (например, импортируемые типы и числа). Метаданные расширяют возможности таких технологий как библиотеки типов и файлы языка описания интерфейсов (IDL). Метаданные всегда связаны с файлом и с IL кодом. Фактически они встроены в exe или dll файлы.

Таким образом, метаданные – это таблицы, в которых есть поля, говорящие о том, что определённый метод находится в определённом файле и принадлежит определённому типу.

Когда впервые происходит запуск программы, сперва происходит анализ заголовка, чтобы узнать какой разрядности процесс запустить. Затем загружается выбранная версия файла MSCorEE.dll. После чего вызывается метод, расположенный в MSCorEE.dll, который инициализирует CLR, сборки и точку входа главной функции программы. Сперва, перед выполнением главной функции программы среда CLR находит все объявленные типы. Затем среда определяет методы, объединяя их в записи внутри единой структуры. Записи содержат адреса, по которым можно найти реализации методов, то есть те преобразования, которые выполняет метод.

В настоящее время при разработке веб-приложений с использованием современных фреймворков существует ряд проблем корректного использования моделей в представлениях. Соответственно необходимо выработать стратегию решения этой проблемы. Для создания программного модуля дипломного проекта в качестве веб-сервиса понадобилось обратиться к использованию архитектуры ASP .NET MVC 4.

MVC (Model View Controller) – это архитектурный паттерн проектирования, который используется для организации программного кода. Его основная идея состоит в том, чтобы отделить модели данных от их отображений и операций взаимодействия с пользователем. Выигрыш от использования такой архитектуры заключается в том, что она позволяет упорядочить код, распределив его по уровням, каждый из которых определяет сферу ответственности. Это минимизирует взаимозависимость программных компонент, что в свою очередь облегчает их последующую модификацию. Доработка и развитие такой системы становится проще.

Пользователь направляет запрос в контроллер, в случае веб-приложений – это обращение по адресу, контроллер (Controller) обрабатывает запрос, запрашивает данные от соответствующих моделей (Model), получает данные, может быть, выполняет какую-то дополнительную их обработку, например, агрегирует их с другими данными и затем передает данные в представление (View). Представление формирует данные в соответствии с заданным шаблоном отображения и возвращает результат пользователю.

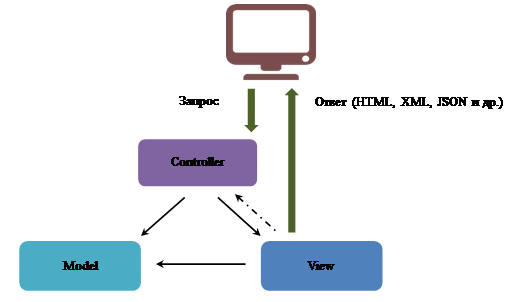


Рисунок 1.7 – Логика работы приложения на основе MVC архитектуры

Внутри функционала контроллеров зачастую присутствуют связи обращения к базе данных. Стратегия запросов к базе заключается в сведении к минимуму или в полном исключении всех SQL-запросов к БД из представления. В рамках этой стратегии инициируются все обращения к БД в контроллере и полученные данные передаются в представление, т.е. представление оперирует данными из оперативной памяти.

Если в представление передается коллекция объектов модели, имеющей агрегированные данные, отражающие реляционную связь с другими моделями, то необходимо осуществить жадную загрузку (eager loading).  Возможность такой загрузки реализована в современных ORM-решениях [6].

Исключение всех операций из представления, инициирующих SQL-запросы – это более строгий по сравнению с первой стратегией подход. Здесь необходимо буквально следовать принципу единичной ответственности в представлении: только отображение имеющихся данных и ничего сверх этого. Методы модели могут быть вызваны только в том случае, если они не генерируют SQL-запросы.

Помимо преимуществ чисто методологического плана - соблюдение принципа единичной ответственности, такой подход привлекателен с точки зрения управления кэшированием данных. В рамках этой стратегии все данные проходят через контроллер, что существенно облегчает проверку их наличия в кэше и сохранения, если данные отсутствуют. Альтернатива – перекладывание этой работы на модель, что, естественно, не самая удачная идея, т.к. ответственность модели неоправданно расширяется. Кэширование же на уровне контроллера вполне естественно и органично.

**1.3** Выбор системы контроля версий

Весь процесс разработки программного кода ведется с использованием программного средства git.

Git — распределённая система управления версиями файлов. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день поддерживается Джунио Хамано.

Примерами проектов, использующих Git, являются ядро Linux, Android, Drupal, Cairo, GNU Core Utilities, Mesa, Wine, Chromium, Compiz Fusion, FlightGear, jQuery, PHP, NASM, MediaWiki и некоторые дистрибутивы Linux (см. ниже).

Программа является свободной и выпущена под лицензией GNU GPL версии 2.

Система спроектирована как набор программ, специально разработанных с учётом их использования в скриптах. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы. Например, Cogito является именно таким примером фронтенда к репозиториям Git, а StGit использует Git для управления коллекцией патчей.

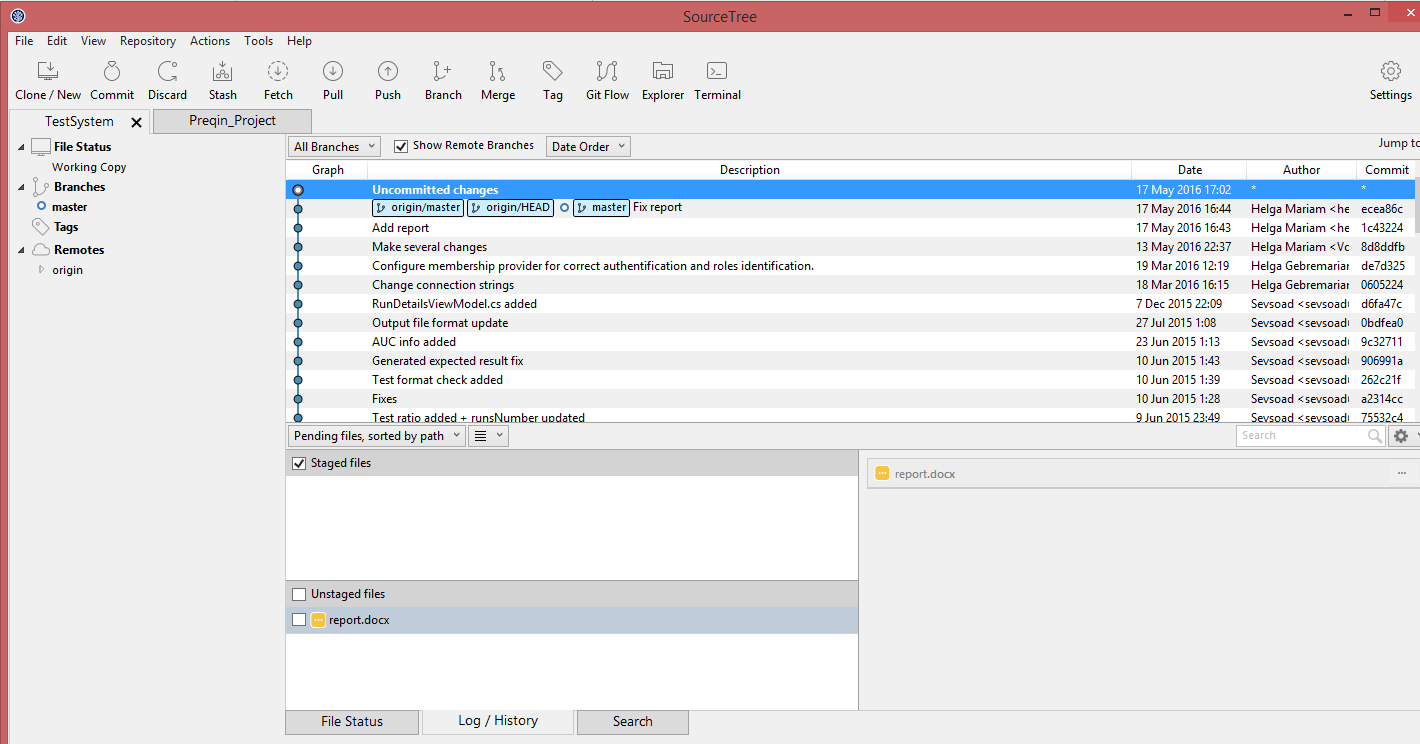


Рисунок 1.8 – Интерфейс системы контроля версий Source Tree

Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone, Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удалённый доступ к репозиториям Git обеспечивается git-daemon, SSH- или HTTPS-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

Ядро Git представляет собой набор утилит командной строки с параметрами. Все настройки хранятся в текстовых файлах конфигурации. Такая реализация делает Git легко портируемым на любую платформу и даёт возможность легко интегрировать Git в другие системы (в частности, создавать графические git-клиенты с любым желаемым интерфейсом).

Репозиторий Git представляет собой каталог файловой системы, в котором находятся файлы конфигурации репозитория, файлы журналов, хранящие операции, выполняемые над репозиторием, индекс, описывающий расположение файлов и хранилище, содержащее собственно файлы. Структура хранилища файлов не отражает реальную структуру хранящегося в репозитории файлового дерева, она ориентирована на повышение скорости выполнения операций с репозиторием. Когда ядро обрабатывает команду изменения (неважно, при локальных изменениях или при получении патча от другого узла), оно создаёт в хранилище новые файлы, соответствующие новым состояниям изменённых файлов. Существенно, что никакие операции не изменяют содержимого уже существующих в хранилище файлов.

По умолчанию репозиторий хранится в подкаталоге с названием «.git» в корневом каталоге рабочей копии дерева файлов, хранящегося в репозитории. Любое файловое дерево в системе можно превратить в репозиторий git, отдав команду создания репозитория из корневого каталога этого дерева (или указав корневой каталог в параметрах программы). Репозиторий может быть импортирован с другого узла, доступного по сети. При импорте нового репозитория автоматически создаётся рабочая копия, соответствующая последнему зафиксированному состоянию импортируемого репозитория (то есть не копируются изменения в рабочей копии исходного узла, для которых на том узле не была выполнена команда commit).

Преимущества и недостатки git по сравнению с централизованными системами управления версиями (такими как, например, Subversion) типичны для любой распределённой системы и описаны в статье «Система управления версиями». Если же сравнивать git с «родственными» ей распределёнными системами, можно отметить, что git изначально идеологически ориентирован на работу с изменениями, а не с файлами, «единицей обработки» для него является набор изменений, или патч. Эта особенность прослеживается как в структуре самой системы (в частности — в структуре репозитория), так и в принципах построения команд; она отражается на производительности системы в различных вариантах её использования и на достоинствах и недостатках git по сравнению с другими системами контроля версий.

Часто называемые преимущества git перед другими DVCS:

- высокая производительность;

- развитые средства интеграции с другими системами контроля версий, в частности, с CVS, SVN и Mercurial. Помимо разнонаправленных конвертеров репозиториев, имеющиеся в комплекте программные средства позволяют разработчикам использовать git при размещении центрального репозитория в SVN или CVS, кроме того, git может имитировать cvs-сервер, обеспечивая работу через клиентские приложения и поддержку в средах разработки, специально не поддерживающих git.

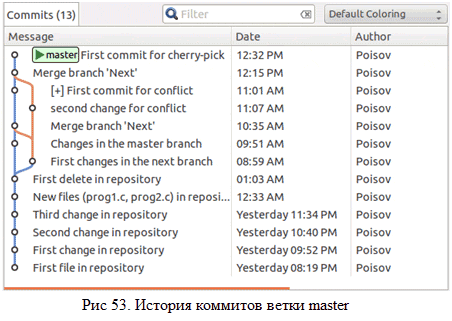


Рисунок 1.8 – Интерфейс системы контроля версий Git

- продуманная система команд, позволяющая удобно встраивать git в скрипты;

- репозитории git могут распространяться и обновляться общесистемными файловыми утилитами архивации и обновления, такими как rsync, благодаря тому, что фиксации изменений и синхронизации не меняют существующие файлы с данными, а только добавляют новые (за исключением некоторых служебных файлов, которые могут быть автоматически обновлены с помощью имеющихся в составе системы утилит). Для раздачи репозитория по сети достаточно любого веб-сервера.

**2** СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Программное средство TestSystem является достаточно эффективной для оценки качества алгоритмов машинного обучения.

Приложение имеет многослойную архитектуру. Данный подход подразумевает разделение ответственности за определенный функционал на соответствующие слои [7]. Например, ответственность слоя DAL (Data Access Layer), как правило, ориентирована на взаимодействие с источником данных (чтение, запись).

Как и любая качественно спроектированная система, Test System позволяет расширять свой функционал, добавлять новые слои.

Задачей данного дипломного проекта является доработка приложения TestSystem с целью улучшения разнообразия его функционала.

Для более качественной оценки методов классификации будет полезно добавить дополнительные алгоритмы тестирования. Это даст разработчикам возможность более полной проверки собственных алгоритмов.

Автоматический генератор тестовых данных позволит пользователю экономить время на ручную генерацию.

Для возможности дальнейшей разработки приложения Test System, чтобы исключить при этом поломку существующего функционала, приложение необходимо покрыть Unit-тестами как можно полнее.

Для решения этих задач были выделены блоки, представленные на структурной схеме (см. чертеж ГУИР.400201.006 С1).

База данных. Модуль, позволяющий хранить данные на жестком диске. Содержит информацию о пользователях системы, сохраненные алгоритмы, а также результаты запуска тестов.

Для реализации хранения этой информации используется система управления базами данных Microsoft SQL Server, которая является высокопроизводительной и эффективной при разработке размером от персональных до крупных баз данных масштаба предприятия.

Для связи таблиц базы данных приложения с уровнем доступа к ним использована технология object-relational mapping.

Основной реализацией данной технологии на платформе .NET является Entity Framework.

Entity Framework предполагает три возможных способа взаимодействия с базой данных:

Database first: Entity Framework создает набор классов, которые отражают модель конкретной базы данных

Model first: сначала разработчик создает модель базы данных, по которой затем Entity Framework создает реальную базу данных на сервере.

Code first: разработчик создает класс модели данных, которые будут храниться в базе данных, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы.

Наиболее корректным подходом в случае с программным средством Test System является подход Database first, поскольку в приложении имеется Database Project, разворачивающий базу данных на локальном сервере.

Database Project является очень удобным решением с точки зрения контроля версий на этапе разработки. Выгода заключается в том, что не приходится каждый раз скачивать файлы базы данных, которые могут достигать очень крупных размеров. Проект содержит описание схемы данных в виде SQL запросов. Для обновления или добавления строк в таблицы базы данных можно использовать, например, Post-deployment scripts, то есть SQL запросы, которые отрабатывают после разворачивания DB Project.

Благодаря DB Project имеется возможность легко сопровождать существующую модель данных, добавлять новые таблицы. Планируется создание Post-Deployment SQL скриптов, заполняющих базу данными, необходимыми для разворачивания приложения на сервере и просмотра его функционала.

Репозиторий пользователей. Блок доступа к данным пользователей системы. Извлекает информацию о данных аутентификации, ролях, а также пользовательских настроек.

В приложении Test System проверка аутентификации, хэширование пароля, заполнение таблиц базы данных информацией о пользователях при регистрации происходит в контроллерах. Такой метод является некорректным с точки зрения многослойной архитектуры. Поэтому в рамках данного дипломного проекта часть функционала будет вынесена из контроллеров в слои бизнес логики и доступа к данным. Для реализации этого будет создан дополнительный блок приложения - блок аутентификации.

Будет использован класс платформы .NET System.WebSecurity, которая обеспечивает защиту и проверку подлинности в приложениях на основе веб-страниц ASP.NET, включая возможность создания учетных записей пользователей, входа и выхода, сброса или изменения паролей, а также выполнения связанных с этим задач.

Алгоритм хэширования пароля необходимо сделать более эффективным и вынести его в слой бизнес логики.

Повторное хэширование будет выполнено методами класса WebSecurity.

Репозиторий тестов. Осуществляет взаимодействие с таблицами, содержащими информацию о результатах запусков тестов.

В эти таблицы будут добавлены дополнительные колонки, содержащие информацию об используемом в данном тесте алгоритме тестирования.

Репозиторий алгоритмов. Реализует считывание и запись данных пользовательских алгоритмов.

Блок автоматической генерации тестов будет создан как часть бизнес логики проекта Test System. Позволит пользователю генерировать тестовые данные для разработанных ими алгоритмов.

Блок генерации псевдослучайных чисел - программная подсистема, реализующая качественную генерацию случайных чисел. Данный ГПСЧ использует новый алгоритм, основанный на синтезе математической базы модифицированного метода Фибоначчи и алгоритма Лемера. Планируется многогранная проверка его эффективности, а также добавление подмодуля, позволяющего задавать в качестве аргументов параметры псевдослучайных значений.

Блок алгоритмов тестирования содержит иерархию классов, оценивающих качество пользовательских алгоритмов.

Блок front-end логики. Оформление программного средства TestSystem выполнено на основе библиотеки Bootstrap.

Bootstrap – это свободный набор инструментов для создания [сайтов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B0%D0%B9%D1%82) и [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5). Включает в себя [HTML](https://ru.wikipedia.org/wiki/HTML) и [CSS](https://ru.wikipedia.org/wiki/CSS) шаблоны оформления для [типографики](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0), веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-расширения.

Библиотека Bootstrap имеет широкий ряд преимуществ:

1. Экономия времени — Bootstrap позволяет сэкономить время и усилия, используя шаблоны дизайна и классы, и сконцентрироваться на других разработках;
2. Высокая скорость — динамичные макеты Bootstrap масштабируются на разные устройства и разрешения экрана без каких-либо изменений в разметке;
3. Гармоничный дизайн — все компоненты платформы Bootstrap используют единый стиль и шаблоны с помощью центральной библиотеки. Дизайн и макеты веб-страниц согласуются друг с другом;
4. Простота в использовании — платформа проста в использовании, пользователь с базовыми знаниями HTML и CSS может начать разработку с Twitter Bootstrap;
5. Совместимость с браузерами — Twitter Bootstrap совместим с Mozilla Firefox, Yandex Browser, Google Chrome, Safari, Internet Explorer и Opera;
6. Открытое программное обеспечение — особенность Twitter Bootstrap, которая предполагает удобство использования, посредством открытости исходных кодов и бесплатной загрузки.

Но надо отметить и недостатки использования данной библиотеки. Она содержит большое количество CSS классов для обеспечения web приложению презентабельного вида. Эти классы хранятся в файле, который подключаются к html странице. И вне зависимости от того, какое количество bootstrap классов было применено на данной странице, вместе с ее загрузкой конечному пользователю каждый раз приходится скачивать целую библиотеку.

С целью экономии ресурсов планируется убрать использование Bootstrap. Верстка сайта будет выполнена на основе собственных CSS файлов.

Для получения данных от сервера со стороны клиента будут использованы AJAX запросы.

AJAX – это подход к построению пользовательских веб-приложений, основанный на «фоновом» обмене данными между клиентом и сервером. В результате при необходимости получения данных не приходится перезагружать HTML страницу полностью, следовательно, приложение работает быстрее, снимается значительная часть нагрузки с сервера.

Последовательность обмена данными при использовании AJAX следующая:

1. Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент.
2. [Скрипт](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82) (на языке [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)) определяет, какая информация необходима для обновления страницы.
3. [Браузер](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) отправляет соответствующий запрос на [сервер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80).
4. [Сервер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80) возвращает только ту часть документа, на которую пришёл запрос.
5. [Скрипт](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82) вносит изменения с учётом полученной информации (без полной перезагрузки страницы).

В классической же модели, без применения AJAX, в ответ на клиентский запрос сервер генерирует совершенно новую HTML страницу и отправляет ее клиенту, после чего браузер полностью перезагружает страницу.

Модель взаимодействия клиента с сервером на примере технологии AJAX в сравнении с классической моделью представлена на рисунке 6.1.

AJAX — не самостоятельная технология, а концепция использования нескольких смежных технологий.

Действия с интерфейсом преобразуются в операции с элементами Document Object Model, с помощью которых обрабатываются данные, доступные пользователю, в результате чего представление их изменяется. Здесь же производится обработка перемещений и щелчков мышью, а также нажатий клавиш. Каскадные таблицы стилей обеспечивают согласованный внешний вид элементов приложения и упрощают обращение к DOM-объектам.

Три из этих четырёх технологий — [CSS](https://ru.wikipedia.org/wiki/CSS), DOM и [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) — составляют Dynamic HTML.

Использование Dynamic HTML обозначено на структурной схеме как блок пользовательского WEB интерфейса.

Запрос серверных данных с применением технологии AJAX происходит посредством обращения к контроллеру web приложения.

Блок контроллеров является компонентом MVC (Model View Controller).

Контроллер обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции, то есть отправки клиенту ответных данных.

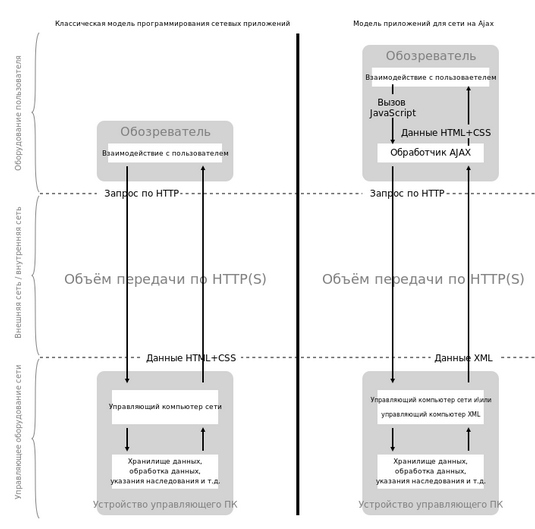


Рисунок 2.1 – Модель классических приложений для сети (слева) в прямом сравнении с применением AJAX (справа).

Итак, в уже существующий проект Test System были добавлены новые функциональные блоки. Также планируется модификация и расширение имеющихся программных компонентов. При этом вся предыдущая функциональность системы будет сохранена.

**3** ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

**3.1** Разработка базы данных

В разрабатываемой базе данных существуют следующие логические модули:

* Модуль пользовательских данных
* Модуль алгоритмов
* Модуль результатов

Модуль пользовательских данных состоит из следующих таблиц:

Users – содержит основную информацию о пользователях, зарегистрированных в системе. В таблице находятся следующие колонки:

Id – авто инкрементируемое поле, содержащее уникальный идентификатор пользователя

UserName – текстовое поле, хранящее имя пользователя

Password – хэш-код пароля

Role – числовое поле, хранит идентификатор роли пользователя; является внешним ключом, ссылающимся на таблицу Roles.

UserInfo – таблица, хранящая дополнительную пользовательскую информацию, такую как Email, полное имя, название компания, а также колонка Other (опциональная дополнительная информация)

Roles – таблица пользовательских ролей. Содержит 2 поля: Id и RoleName.

UserSavedSettings – таблица пользовательских настроек. Хранит следующие поля:

Id – уникальный идентификатор параметра

UserId – идентификатор пользователя, внешний ключ, ссылающийся на таблицу Users

ObjectName – название сохраненного параметра

ObjectValue – значение сохраненного параметра

Модуль алгоритмов состоит из одной таблицы – Algorithms.

Таблица содержит следующие колонки:

Id – уникальный идентификатор с автоинкрементом

CreatorId – внешний ключ, ссылающийся на таблицу Users

Description – текстовое описание алгоритма

Name – название алгоритма

DateOfCreation – дата создание алгоритма. Поле автоматически заполняется текущей датой на момент сохранения записи в поле.

Модуль результатов состоит таблиц TestRuns и TestResults.

Таблица TestRuns содержит информацию о запусках тестирования конкретных тестовых наборов с конкретными алгоритмами. Поле DateOfRun хранит информацию о запуске теста, имеет тип datetime. AlgorithmId является внешним ключом к таблице Algorithms, образуя связь один ко многим. Поле TestCollectionId является внешним ключом к таблице TestDataSets, указывая с каким тестовым набором проводится запуск тестирования. Таким образом с этой таблицей образуется связь один ко многим. Поле UserId является внешним ключом к таблице Users, указывая какой пользователь производил запуск тестирования и образуя с этой таблицей связь один ко многим, так как один пользователь может производить множество запусков тестирования. RunComments имеет тип nvarchar максимального размера и хранит дополнительную информацию от тестировании. Это поле может хранить информацию о параметрах используемых при обучении и распознавании, также в этом поле могут храниться координаты для построения рок-кривой, по проведенному тестированию. В этом случае указывается варьируемый параметр.

Развернуть текущую модель данных на сервере позволяет Database Project. Он хранит в себе таблицы базы данных в виде SQL скриптов (рис. 3.1)

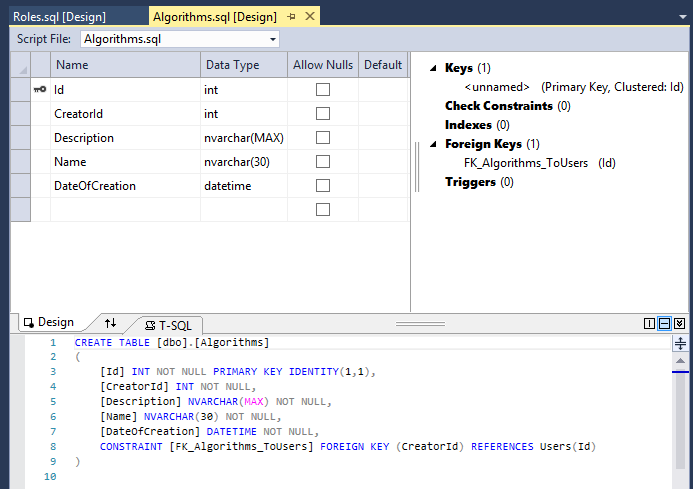


Рисунок 3.1 – Хранение таблиц в Database Project

Database Project является удобным подходом в случае, когда в проекте используется тяжеловесная база данных, а также при командной разработке. Такой подход значительно облегчает контроль версий, так как дает возможность наглядно увидеть, каким изменениям была подвержена схема, а затем накатить эти изменения на свою локальную базу данных.

Таким образом, для контроля версий нет необходимости каждый раз скачивать измененную версию файлов базы данных, достаточно лишь применить внесенные изменения.

Также нет необходимости использовать в процессе разработки общую базу данных. Это позволяет избежать некорректной работы, связанной со случайными действиями разработчиков, а в случае необходимости откатить внесенные неверные изменения.

При этом существует два способа применить изменения на своей локальной базе данных. Способ первый – посредством функции «Schema Compare» сравнить 2 схемы данных, а затем обновить целевую схему (рисунок 3.2).

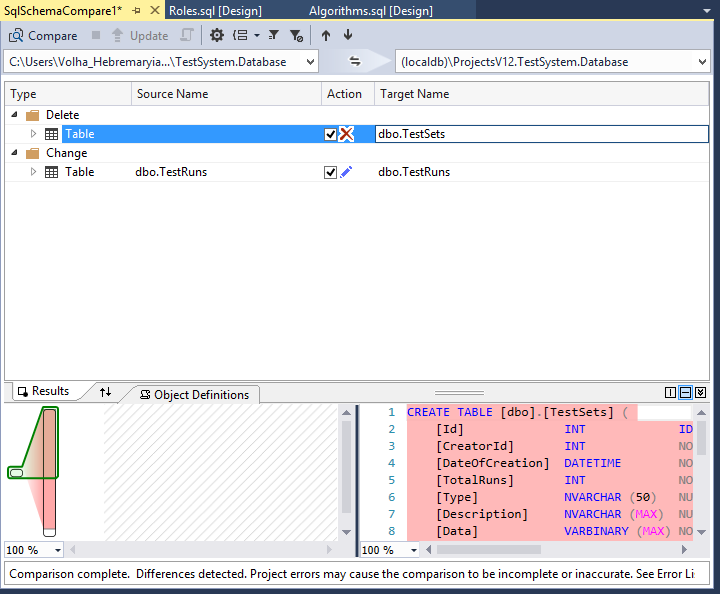


Рисунок 3.2 – Сравнение схем данных посредством Visual Studio

Способ второй – функция Publish, позволяющая либо накатить все изменения на целевую базу данных, либо создать скрипт, применяющий эти изменения (рисунок 3.3).

Преимущество этого способа заключается в том, что в данном случае не только происходит обновление схемы целевой базы данных, но и отрабатывают Pre-deployment скрипты перед обновлением, а затем, после того, как произошел Publish, отрабатывают Post-deployment скрипты. Они представляют собой инструкции на языке SQL.

Это может быть полезно, например, в том случае, когда нам необходимо применить изменения не только на схеме данных, но и переписать либо дополнить сами данные.

Например, в определенной таблице добавился внешний ключ, или же дополнительное поле, требующее заполнения для корректной работы приложения.

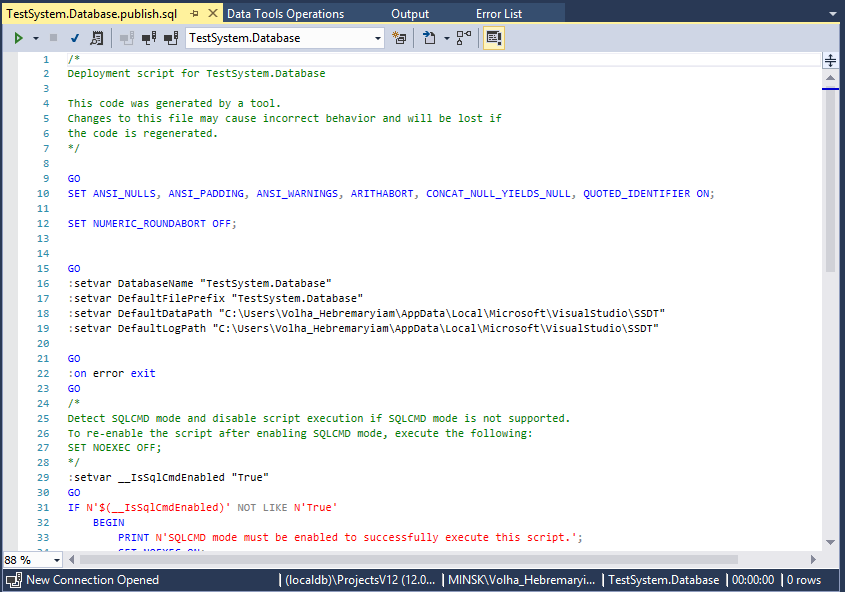


Рисунок 3.3 – Сгенерированный publish script

Для того чтобы приступить к работе с приложением, нам необходимо, чтобы в базе данных был хотя бы один пользователь. А для создания пользователя необходима заполненная таблица Roles.

Поэтому для удобства установки проекта на локальной машине в проект базы данных были добавлены Post-Deployment скрипты.

Скрипт AddRoles.sql содержит инструкции по добавлению данных в таблицу ролей:

IF NOT EXISTS (select \* from Roles where RoleName = 'User')

BEGIN

INSERT INTO Roles

(RoleName)

VALUES

('User')

END

IF NOT EXISTS (select \* from Roles where RoleName = 'Admins')

BEGIN

INSERT INTO Roles

(RoleName)

VALUES

('Admins')

END

А после скрипта AddRoles.sql отрабатывает скрипт AddUser.sql, добавляющий первичного пользователя с правами Admin:

DECLARE @roleID INTEGER

SELECT @roleID = Id from Roles where RoleName = 'Admins'

IF NOT EXISTS (select \* from Users)

BEGIN

INSERT INTO Users

(Password, Role, UserName)

VALUES

('39BCC8974196C14EC16627CE65832820', @roleID, 'User')

END

Как видно, в sql инструкциях осуществляется проверка на существование записей в базе данных. В случае, если в таблице ролей базовые роли, такие как Admins и User, существуют, запись не происходит.

Что касается таблицы Users, то первичный пользователь добавляется только в том случае, если таблица пуста.

Таким образом, скрипты могут отрабатывать больше одного количества раз без ущерба существующим данным.

**3.2** Разработка уровня доступа к данным

Для доступа к данным через приложение TestSystem была выбрана платформа ADO.NET Entity Framework, позволяющая разработчикам не иметь напрямую дела с реляционной схемой хранения данных, а работать с концептуальной моделью приложения.

Данная платформа имеет широкий ряд преимуществ, таких как:

* Приложения могут работать концептуальной моделью в терминах предметной области — в том числе с наследуемыми типами, сложными элементами и связями.
* Приложения освобождаются от жестких зависимостей от конкретного ядра СУБД или схемы хранения.
* Сопоставления между концептуальной моделью и схемой, специфичной для конкретного хранилища, могут меняться без изменения кода приложения.
* Разработчики имеют возможность работать с согласованной моделью объектов приложения, которая может быть сопоставлена с различными схемами хранения, которые, возможно, реализованы в различных системах управления данными.
* Несколько концептуальных моделей могут быть сопоставлены с единой схемой хранения.
* Поддержка запросов LINQ обеспечивает проверку синтаксиса во время компиляции для запросов к концептуальной модели.

Поскольку база данных уже готова и развернута на локальном сервере, доступ к данным через приложение будет осуществляться с использованием подхода Database First.

Работа непосредственно с данными, то есть обращения к таблицам базы из приложения при этом подходе ничем не отличается от подхода Code First. Разница существует лишь в процессе создания слоя ORM (Object-Relational Mapping, или же объектно-реляционное отображение).

Изначально в среде Visual Studio был создан проект библиотеки классов, который будет представлять из себя слой ORM, а затем добавлен объект ADO.NET Entity Data Model (рисунок 3.4).

Далее было выбрано подключение к базе данных, развернутой на локальном сервере. И затем после компиляции приложения был автоматически сгенерирован класс контекста для кода доступа к базе данных.

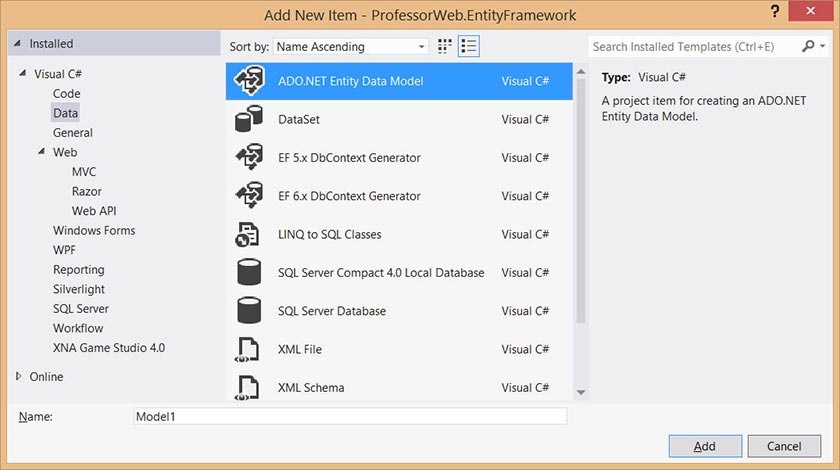


Рисунок 3.4 – Создание ADO.NET Entity Data Model

**3.3** Разработка генератора тестовых наборов данных

Оценка качества классификаторов будет происходить на основе их способности к тестированию тестовых наборов данных.

Для удобства использования будет предоставлена возможность генерации тестовых наборов данных, то набора случайных чисел нужной длины, с заданными вероятностными характеристики.

Количество случайных чисел, требуемых для получения статистически устойчивых оценок параметров процессов функционирования системы при реализации моделирующего алгоритма, как правило, колеблется в широких пределах в зависимости от класса объекта моделирования, вида оцениваемых параметров, необходимой точности и достоверности результатов.

Для метода статистического моделирования на ЭВМ характерно то, что большое число операций, соответственно и большая доля машинного времени расходуется на действия со случайными числами. Кроме того, результаты статистического моделирования существенно зависят от качества исходных последовательностей случайных чисел. Поэтому наличие простых и экономичных способов формирование последовательностей случайных чисел требуемого качества во многом определяет возможности практического использования машинного моделирования систем.

Алгоритмический способ получения последовательностей случайных чисел в настоящее время считается наиболее эффективным. Он основан на формировании случайных чисел с помощью специальных алгоритмов и реализующих их программ. Каждое случайное число вычисляется с помощью соответствующего алгоритма по мере возникновения необходимости.

При моделировании качественного генератора псевдослучайных чисел программная имитация случайных воздействий любой сложности сводится к генерированию некоторых базовых последовательностей случайных чисел и их последующему функциональному преобразованию. В качестве базового может быть принят любой удобный процесс.

Как показывает практика, оптимальным базовым процессом является последовательность чисел {X}= x1,x2,…,xn, представляющих собой реализацию равномерно распределенной на интервале (0,1) случайной величины ξ, или - в статистических терминах - повторную выборку из равномерно распределенной на (0,1) генеральной совокупности значений величины ξ.

Непрерывная случайная величина ξ имеет равномерное распределение в интервале (a,b), если ее функции плотности fξ(x) и функция распределения Fξ(x) соответственно имеют вид:

Далее на основе равномерно распределенной последовательности чисел имеется возможность получить последовательность с заданным распределением на основе математических формул.

Генератор псевдослучайных чисел программного модуля TestSystem позволяет пользователю получить последовательность со следующими видами распределений.

Гауссовское распределение (рисунок 3.5)

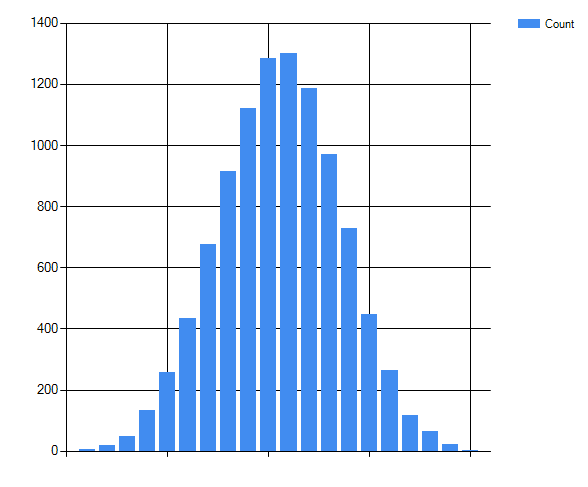


Рисунок 3.5 – Гауссовское распределение

Гауссовское распределение является одним из наиболее распространенных непрерывных распределений. Гауссовская аппроксимация реального распределения используется обычно в следующих случаях:

1) когда реальное распределение обусловлено теми факторами, которые определяются центральной предельной теоремой теории вероятности;

2) когда реальное распределение известно, однако допускается его гауссовская аппроксимация с целью упрощения решаемой задачи;

3) когда реальное распределение неизвестно, однако нет каких-либо оснований отвергать его гауссовскую аппроксимацию.

Гауссовское распределение непрерывной случайной величины X описывается плотностью распределения:

где mx и σx - соответственно математическое ожидание и среднее квадратическое отклонение гаусовского распределения.

Машинный алгоритм для имитации гауссовского распределения можно получить, базируясь на центральной предельной теореме. Эта теорема утверждает, что сумма независимых, случайных величин с произвольными распределениями имеет асимптотически гауссовское распределение. Сходимость к гауссовскому распределению осуществляется наиболее быстро, если суммируются величины с одинаковым распределением. В этом случае даже небольшое число слагаемых приводит к гауссовскому распределению.

В основе машинного алгоритма для имитации гауссовского распределения лежит суммирование случайных р.р. чисел R.

С возрастанием n, т.е. числа суммируемых случайных р.р. чисел R, повышается точность имитации гауссовского распределения. Обычно n выбирают в пределах от 6 до 12. При этом достаточная для многих приложений точность обеспечивается при использовании всего шести случайных р.р. чисел R. Для случая, когда n = 6,

Формула (11) представляет собой искомый машинный алгоритм, который наиболее часто используется на практике. С помощью этого алгоритма имитируется гауссовская случайная величина x с заданным статистическими параметрами mx и σx.

Экспоненциальное распределение (рисунок 3.6)

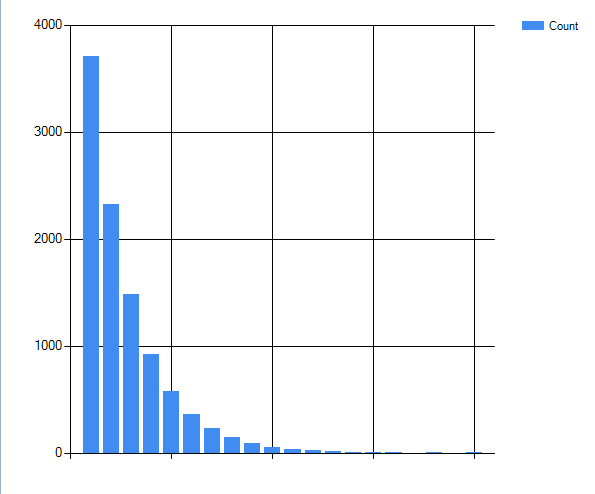


Рисунок 3.6 – Гистограмма экспоненциального распределения

Экспоненциальное распределение непрерывной случайной величины X описывается плотностью распределения

Где λ - параметр экспоненциального распределения.

Математическое ожидание и дисперсия случайной величины X определяются соотношениями

Получим машинный алгоритм для имитации экспоненциального распределения, используя метод обратных функций:

(12)

Формула (12) представляет собой искомый машинный алгоритм.

Гамма распределение (рисунок 3.7)

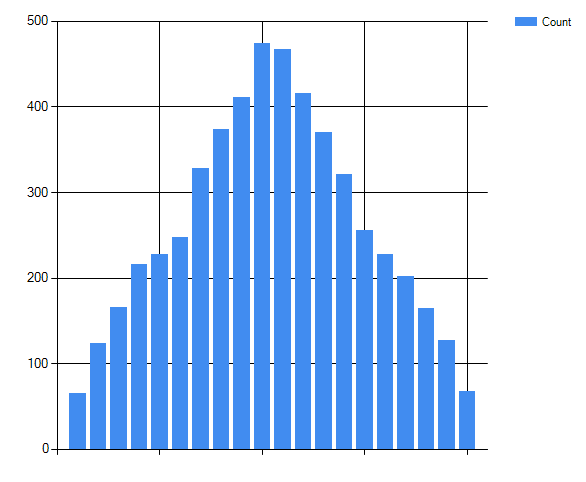


Рисунок 3.7 – Гистограмма гамма распределения

Гамма-распределение непрерывной случайной величины  описывается плотностью распределения:

где η и λ - параметры гамма-распределения (η>0, λ>0)).

При η, принимающем целочисленные значения, гамма-распределение называется распределением Эрланга.

Математическое ожидание и дисперсия случайной величины x определяются соотношениями

Гамма-распределение сводится к экспоненциальному распределению, если положить η =1. Случайная величина x может быть представлена в виде суммы независимых случайных величин xi, имеющих экспоненциальное распределение:

Получим машинный алгоритм для имитации гамма-распределения:

или

(13)

Где случайные р. р. числа.

Треугольное распределение (рисунок 3.8)

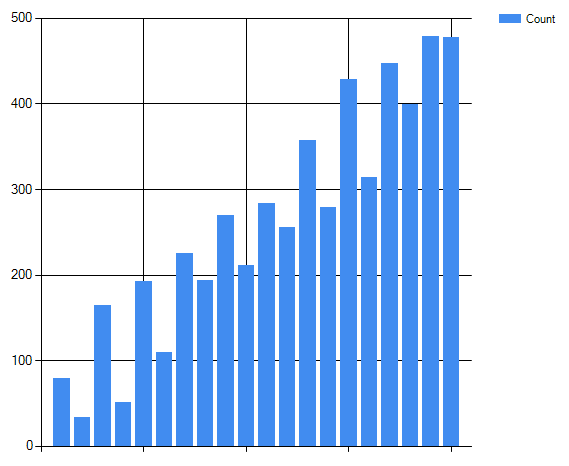


Рисунок 3.8 – Гистограмма треугольного распределения

Треугольное распределение непрерывной случайной величины описывается плотностями распределения:

или

Для имитации треугольного распределения может быть использован метод исключения, предложенный И.Нейманом.

Машинный алгоритм для имитации треугольного распределения с плотностью (14):

1. Формируются пара случайных р. р. чисел R1 и R2.

2. Проверяется условие R1 < R2. Если условие выполняется, то искомое число

В противном случае, пара чисел R1 и R2 отбрасывается и осуществляется переход к шагу 1.

Машинный алгоритм для имитации треугольного распределения с плотностью (15).

1. Формируются два случайных р. р. числа R1 и R2.

2. Проверяется условие . Если условие выполняется, то находится искомое число .

В противном случае пара чисел R1 и R2 отбрасывается и осуществляется переход к шагу 1.

Приведенные алгоритмы имеют существенный недостаток: часть пар чисел R1 и R2, приходится отбрасывать. Принимая во внимание независимость случайных р. р. чисел R1 и R2, можно предложить более экономичные алгоритмы, основанные на использовании следующих формул:

где

max(R1, R2 ) - взятие максимального числа из совокупности двух случайных р. р. чисел R1 и R2;

min(R1, R2 ) - взятие минимального числа на совокупности двух случайных р. р. чисел R1 и R2.

Формулы (16) и (17) представляют собой машинные алгоритмы для имитации треугольного распределения с плотностями соответственно (14) и (15).

Распределение Симпсона (рисунок 3.9)

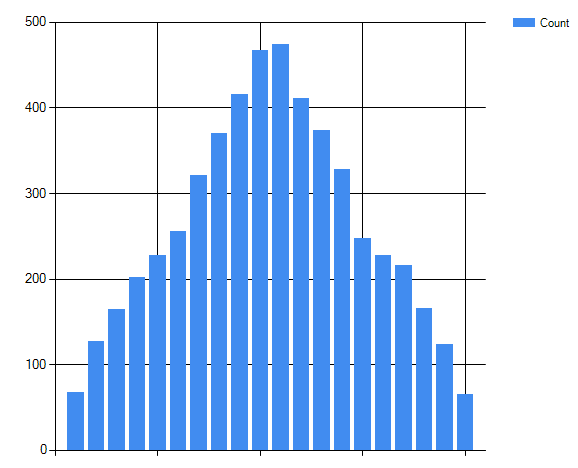


Рисунок 3.8 – Гистограмма распределение Симпсона

Распределение Симпсона непрерывной случайной величины  описывается плотностью распределения

Распределение Симпсона имеет случайная величина , которая представляет собой следующую сумму:

где y и z - независимые случайные величины, распределенные равномерно на интервале . Следовательно, распределение Симпсона можно рассматривать как композицию двух одинаковых законов равномерного распределения.

Машинный алгоритм для имитации распределения Симпсона базируется на применении формулы (18). Согласно этой формуле необходимо получить два случайных числа y и z , распределенные равномерно на интервале , и просуммировать их. Найденное таким образом число X будет иметь распределение Симпсона.

**4** РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

**4.1** Описание алгоритма хэширования паролей

Хеширование паролей является одним из самых базовых соображений безопасности, которые необходимо сделать при разработке приложения, принимающего пароли от пользователя.

Без хэширования пароли, хранящиеся в базе приложения, могут быть украдены, например, если база данных была скомпрометирована, а затем немедленно могут быть применены для компрометации не только приложения, но и аккаунтов пользователей на других сервисах, если они не используют уникальных паролей.

При применении хеширующего алгоритма к пользовательским паролям перед сохранением их в своей базе данных, разгадывание оригинального пароля становится невозможным для атакующего базу данных. При этом сохраняется возможность сравнения полученного хеша с оригинальным паролем.

Важно заметить, что хеширование паролей защищает их только от компрометирования в хранилище, но необязательно от вмешательства вредоносного кода в приложении.

Также необходимо отметить, что для текущих целей не подойдут популярные хеширующие алгоритмы, несмотря на то, что они были спроектированы очень быстрыми и эффективными. При наличии современных технологий и оборудования стало довольно просто выяснить результат этих алгоритмов методом «грубой силы» для определения оригинальных вводимых данных.

Из-за той скорости, с которой современные компьютеры могут «обратить» эти хеширующие алгоритмы, многие профессионалы компьютерной безопасности строго не рекомендуют использовать их для хеширования паролей.

При хешировании паролей существует два важных соображения: это стоимость вычисления и соль. Чем выше стоимость вычисления хеширующего алгоритма, тем больше времени требуется для взлома его методом «грубой силы».

Возможности языка C# предоставляют стандартные методы хеширования. Описаны эти методы в пространстве имен System.Security.Cryptography. В программном модуле TestSystem методы класса MD5 были взяты за основу для расчета хеш кода.

Целью данного программного модуля является создание трудно просчитываемого алгоритма, отвечающего основным критериям качества.

Хорошая хеш-функция — должна удовлетворять двум свойствам:

* Быстрое вычисление;
* Минимальное количество «[коллизий](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%BB%D0%B8%D0%B7%D0%B8%D1%8F_%D1%85%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8)»

Коллизией хеш-функции называются такие два вводимых блока данных, которые дают одинаковые хеш-коды.

Существует несколько способов для защиты от подделки паролей и [подписей](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C), работающих даже в том случае, если криптоаналитику известны способы построения коллизий для используемой хэш-функции. Одним из таких методов является добавление [криптографической соли](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BB%D1%8C_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) (строки случайных данных) к входным данным (иногда «соль» добавляется и к хэш-коду), что значительно затрудняет анализ итоговых хэш-таблиц.

Криптографическая соль представляет собой данные, которые применяются в процессе хеширования для предотвращения возможности разгадать оригинальный ввод с помощью писка результата хеширования в списке заранее вычисленных пар ввод-хеш, известном также как «радужная» таблица.

«Соль» - это часть дополнительных данных, которые делают хеши намного более устойчивыми к взлому. Существует много онлайн-сервисов, предоставляющих обширные списки заранее вычисленных хешей вместе с их оригинальным вводом. Использование соли делает поиск результирующего хеша в таком списке маловероятным и даже невозможным.

Среди множества существующих хэш-функций принято выделять [криптографически стойкие](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D1%82%D0%BE%D0%B9%D0%BA%D0%BE%D1%81%D1%82%D1%8C" \o "Криптографическая стойкость), применяемые в [криптографии](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F), так как на них накладываются дополнительные требования.

Для того чтобы хеш-функция H считалась криптографически стойкой, она должна удовлетворять трём основным требованиям, на которых основано большинство применений хеш-функций в криптографии:

Необратимость: для заданного значения хеш-функции m должно быть вычислительно неосуществимо найти блок данных X, для которого H(X) = m.

Стойкость к коллизиям первого рода: для заданного сообщения M должно быть вычислительно неосуществимо подобрать другое сообщение N, для которого H(N) = H(M).

Стойкость к коллизиям второго рода: должно быть вычислительно неосуществимо подобрать пару сообщений (M, M’), имеющих одинаковый хеш.

Данные требования не являются независимыми:

Обратимая функция нестойка к коллизия первого и второго рода

Функция, нестойкая к коллизиям первого рода, нестойка к коллизиям второго рода; обратное неверно.

Следует отметить, что не доказано существование необратимых хэш-функций, для которых вычисление какого-либо прообраза заданного значения хэш-функции теоретически невозможно. Обычно нахождение обратного значения является лишь вычислительно сложной задачей.

Атака «дней рождения» позволяет находить коллизии для хеш-функции с длиной значений n битов в среднем за примерно 2n/2 вычислений хеш-функции. Поэтому n-битная функция считается криптостойкой, если вычислительная сложность нахождения коллизий для нее близка к 2n/2.

Для криптографических хэш-функций также важно, чтобы при малейшем изменении аргумента значение функции сильно изменялось ([лавинный эффект](https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D0%B2%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82)). В частности, значение хеша не должно давать утечки информации даже об отдельных [битах](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82) аргумента. Это требование является залогом криптостойкости алгоритмов хэширования, хеширующих пользовательский пароль для получения [ключа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)).

MD5 ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Message Digest 5) — 128-битный алгоритм [хеширования](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), разработанный профессором [Рональдом Л. Ривестом](https://ru.wikipedia.org/wiki/%D0%A0%D0%B8%D0%B2%D0%B5%D1%81%D1%82,_%D0%A0%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%B4)из [Массачусетского технологического института](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B0%D1%87%D1%83%D1%81%D0%B5%D1%82%D1%81%D0%BA%D0%B8%D0%B9_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%81%D1%82%D0%B8%D1%82%D1%83%D1%82) (Massachusetts Institute of Technology, MIT) в [1991 году](https://ru.wikipedia.org/wiki/1991_%D0%B3%D0%BE%D0%B4). Предназначен для создания «отпечатков» или [дайджестов сообщения](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%B9%D0%B4%D0%B6%D0%B5%D1%81%D1%82_%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F) произвольной длины и последующей проверки их подлинности. Широко применялся для проверки [целостности информации](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8) и хранения паролей в закрытом виде.

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения может быть любой (в том числе нулевой). Запишем длину сообщения в L. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

Шаг 1. Выравнивание потока[[править](https://ru.wikipedia.org/w/index.php?title=MD5&veaction=edit&vesection=3" \o "Редактировать раздел «Шаг 1. Выравнивание потока») | [править вики-текст](https://ru.wikipedia.org/w/index.php?title=MD5&action=edit&section=3)]

Сначала дописывают единичный бит в конец потока (байт 80h), затем необходимое число нулевых бит. Входные данные [выравниваются](https://ru.wikipedia.org/wiki/%D0%94%D0%BE%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) так, чтобы их новый размер L’ был [сравним](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8E_%D0%BD%D0%B0%D1%82%D1%83%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D1%87%D0%B8%D1%81%D0%BB%D0%B0) с 448 по модулю 512, (L’=512\*N+448). Выравнивание происходит, даже если длина уже сравнима с 448.

Шаг 2. Добавление длины сообщения

В конец сообщения дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта, затем старшие. Если длина превосходит 264-1, то дописывают только младшие биты (эквивалентно взятию по модулю 264). После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

Шаг 3. Инициализация буфера

Для вычислений инициализируются 4 переменных размером по 32 бита и задаются начальные значения шестнадцатеричными числами (порядок байтов [little-endian](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2" \o "Порядок байтов), сначала младший байт):

А = 01 23 45 67; // 67452301h

В = 89 AB CD EF; // EFCDAB89h

С = FE DC BA 98; // 98BADCFEh

D = 76 54 32 10. // 10325476h

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD называется инициализирующим вектором.

Определим ещё функции и константы, которые нам понадобятся для вычислений.

Потребуются 4 функции для четырёх раундов. Введём функции от трёх параметров — слов, результатом также будет слово:

1-й раунд:

2-й раунд:

3-й раунд:

4-й раунд:

Определим таблицу констант T[1…64] – 64-элементная таблица данных, построенная следующим образом:

Каждый 512-битный блок проходит 4 этапа вычислений по 16 раундов. Для этого блок представляется в виде массива X из 16 слов по 32 бита. Все раунды однотипны и имеют вид: [abcd k s i], определяемый как , где k – номер 32-битного слова из текущео 512-битного блока сообщения и … <<<< s – циклический сдвиг влево на s бит полученного 32-битного аргумента.

Число s задается отдельно для каждого раунда.

Шаг 4. Вычисление в цикле

Заносим в блок данных элемент n из массива 512-битных блоков. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

AA = A

BB = B

CC = C

DD = D

Этап 1

/\* [abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 0 7 1][DABC 1 12 2][CDAB 2 17 3][BCDA 3 22 4]

[ABCD 4 7 5][DABC 5 12 6][CDAB 6 17 7][BCDA 7 22 8]

[ABCD 8 7 9][DABC 9 12 10][CDAB 10 17 11][BCDA 11 22 12]

[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]

Этап 2

/\* [abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 1 5 17][DABC 6 9 18][CDAB 11 14 19][BCDA 0 20 20]

[ABCD 5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA 4 20 24]

[ABCD 9 5 25][DABC 14 9 26][CDAB 3 14 27][BCDA 8 20 28]

[ABCD 13 5 29][DABC 2 9 30][CDAB 7 14 31][BCDA 12 20 32]

Этап 3

/\* [abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 5 4 33][DABC 8 11 34][CDAB 11 16 35][BCDA 14 23 36]

[ABCD 1 4 37][DABC 4 11 38][CDAB 7 16 39][BCDA 10 23 40]

[ABCD 13 4 41][DABC 0 11 42][CDAB 3 16 43][BCDA 6 23 44]

[ABCD 9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA 2 23 48]

Этап 4

/\* [abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). \*/

[ABCD 0 6 49][DABC 7 10 50][CDAB 14 15 51][BCDA 5 21 52]

[ABCD 12 6 53][DABC 3 10 54][CDAB 10 15 55][BCDA 1 21 56]

[ABCD 8 6 57][DABC 15 10 58][CDAB 6 15 59][BCDA 13 21 60]

[ABCD 4 6 61][DABC 11 10 62][CDAB 2 15 63][BCDA 9 21 64]

Суммируем с результатом предыдущего цикла:

A = AA + A

B = BB + B

C = CC + C

D = DD + D

После окончания цикла необходимо проверить, есть ли ещё блоки для вычислений. Если да, то переходим к следующему элементу массива (n + 1) и повторяем цикл.

Шаг 5. Результат вычислений

Результат вычислений находится в буфере ABCD, это и есть хеш. Если выводить побайтово, начиная с младшего байта A и закончив старшим байтом D, то мы получим MD5-хеш. 1, 0, 15, 34, 17, 18…

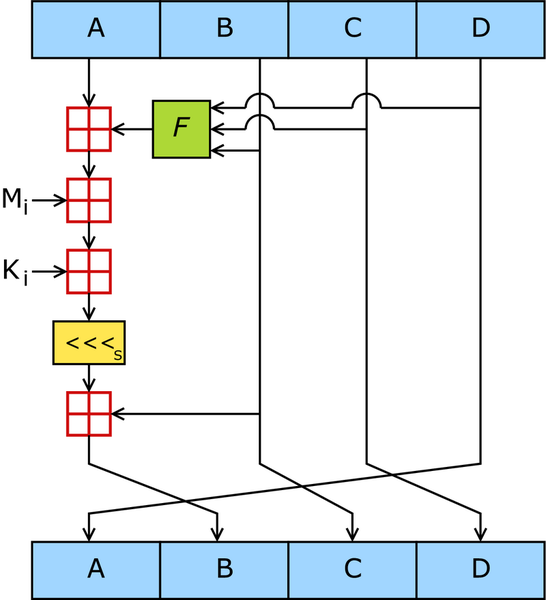


Рисунок 4.1 – Схема работы алгоритма MD5

**4.2** Описание алгоритма регистрации и авторизации пользователей

В языке C# существует встроенная система авторизации, которая называется AspNetMembershipProvider.

Однако данный подход не является оптимальным с точки зрения производительности системы.

Например, если рассмотреть файлы InstallMembership.sql и InstallProfile.sql, то sql код для FindUsersByName будет выглядеть следующим образом:

CREATE PROCEDURE dbo.aspnet\_Membership\_FindUsersByName @ApplicationName nvarchar(256),

@UserNameToMatch nvarchar(256),

@PageIndex int,

@PageSize int

AS

BEGIN

DECLARE

@ApplicationId uniqueidentifier

SELECT @ApplicationId = NULL

SELECT @ApplicationId = ApplicationId

FROM dbo.aspnet\_Applications

WHERE LOWER(@ApplicationName) = LoweredApplicationName

IF (@ApplicationId IS NULL) RETURN 0

DECLARE @PageLowerBound int

DECLARE @PageUpperBound int

DECLARE @TotalRecords int

SET @PageLowerBound = @PageSize \* @PageIndex SET @PageUpperBound = @PageSize - 1 + @PageLowerBound -- Create a temp table TO store the select results

CREATE TABLE #PageIndexForUsers ( IndexId int IDENTITY (0, 1) NOT NULL, UserId uniqueidentifier ) -- Insert into our temp table INSERT INTO #PageIndexForUsers (UserId)

SELECT u.UserId FROM dbo.aspnet\_Users u, dbo.aspnet\_Membership m

WHERE u.ApplicationId = @ApplicationId AND m.UserId = u.UserId

AND u.LoweredUserName LIKE LOWER(@UserNameToMatch)

ORDER BY u.UserName

SELECT u.UserName, m.Email, m.PasswordQuestion, m.Comment, m.IsApproved, m.CreateDate, m.LastLoginDate, u.LastActivityDate, m.LastPasswordChangedDate, u.UserId, m.IsLockedOut, m.LastLockoutDate

FROM dbo.aspnet\_Membership m, dbo.aspnet\_Users u, #PageIndexForUsers p

WHERE u.UserId = p.UserId AND u.UserId = m.UserId AND p.IndexId >= @PageLowerBound AND p.IndexId <= @PageUpperBound

ORDER BY u.UserName

SELECT @TotalRecords = COUNT(\*) FROM #PageIndexForUsers

RETURN @TotalRecords END

Как видно, постоянно создаются временные таблицы, что является неоптимальным с точки зрения производительности.

Поэтому в программном средстве Test System для регистрации и авторизации пользователей был написан собственный провайдер, который для получения информации использует cookies.

Cookies – это часть информации, отсылаемая сервером браузеру, которую браузер возвращает обратно серверу вместе с каждым запросом.

Для получения cookie данных используется следующая программная конструкция:

var cookie = Request.Cookies["username"]

Для хранения данных авторизации в Cookies в зашированном виде использован класс FormsAuthenticationTicket.

Был реализован интерфейс IPrincipal и установлен в HttpContext.User для проверки ролей IIdentity интерфейса.

Для интерфейса IIdentity была выполнена реализация.

В UserController создано свойство CurrentUser, которое возвращает значение пользователя, который на данный момент залогинен.

Таким образом, как только в модуль авторизации поступает информация о контексте и содержащихся в нем cookie данных, появляется доступ к пользователю. Таким образом существует возможность получения информации о данном пользователи через репозиторий по требованию.

**4.3** Описание алгоритма генерации случайных чисел

Известны следующие виды алгоритмов генерации случайных чисел:

Аддитивные генераторы

Линейный конгруэнтные алгоритмы

Линейный конгруэнтный метод — один из методов [генерации псевдослучайных чисел](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D0%BF%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB). Применяется в простых случаях и не обладает [криптографической стойкостью](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D1%82%D0%BE%D0%B9%D0%BA%D0%BE%D1%81%D1%82%D1%8C). Входит в стандартные библиотеки различных [компиляторов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80).

Хотя линейный конгруэнтный метод порождает статистически хорошую псевдослучайную последовательность чисел, он не является криптографически стойким. Генераторы на основе линейного конгруэнтного метода являются предсказуемыми, поэтому их нельзя использовать в криптографии. Впервые генераторы на основе линейного конгруэнтного метода были взломаны Джимом Ридсом (Jim Reeds), а затем Джоан Бояр (Joan Boyar). Ей удалось также вскрыть квадратические и кубические генераторы. Другие исследователи расширили идеи Бояр, разработав способы вскрытия любого полиномиального генератора. Таким образом, была доказана бесполезность генераторов на основе конгруэнтных методов для криптографии. Однако генераторы на основе линейного конгруэнтного метода сохраняют свою полезность для некриптографических приложений, например, для моделирования. Они эффективны и в большинстве используемых эмпирических тестах демонстрируют хорошие статистические характеристики.

Запаздывающие генераторы Фибоначчи ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) lagged Fibonacci generator) — [генераторы псевдослучайных чисел](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D0%BF%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D1%81%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB), также называемые аддитивными генераторами.

В отличие от генераторов, использующих [линейный конгруэнтный алгоритм](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BD%D0%B3%D1%80%D1%83%D1%8D%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4), фибоначчиевы генераторы можно использовать в статистических алгоритмах, требующих высокого разрешения.

В связи с этим линейный конгруэнтный алгоритм постепенно потерял свою популярность и его место заняло семейство фибоначчиевых алгоритмов, которые могут быть рекомендованы для использования в алгоритмах, критичных к качеству [случайных чисел](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%BE%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE).

Таким образом, в качестве метода генерации случайных чисел был выбран метод Фибоначчи с запаздываниями.

Один из широко распространённых фибоначчиевых генераторов основан на следующей итеративной формуле:

Где xk – вещественные числа из диапазона [0,1), a,b – целые положительные числа, называемые лагами. При реализации через целые числа достаточно формулы . При этом будут происходить арифметические переполнения.

Для работы фибоначчиеву генератору требуется знать max{a,b} предыдущих сгенерированных случайных чисел. При программной реализации для хранения сгенерированных случайных чисел используется конечная циклическая очередь на базе массива. Для старта фибоначчиевому генератору требуется max{a,b} случайных чисел, которые могут быть сгенерированы простым конгруэнтным методом.

Получаемые случайные числа обладают хорошими статистическими свойствами, причём все биты случайного числа равнозначны по статистическим свойствам. Период фибоначчиева генератора может быть оценён по следующей формуле:

, где l – число битов в мантиссе вещественного числа.

Лаги a и b – «магические» и их не следует выбирать произвольно. В некоторых источниках рекомендуют использование следующих значений лагов: (a,b)=(55,24),(17,5) или (97,33).

Качество получаемых случайных чисел зависит от значения константы, a чем оно больше, тем выше размерность пространства, в котором сохраняется равномерность случайных векторов, образованных из полученных случайных чисел. В то же время, с увеличением величины константы a увеличивается объём используемой алгоритмом памяти.

Значения (a,b)=(17,5) можно рекомендовать для простых приложений, не использующих векторы высокой размерности со случайными компонентами. Значения (a,b)=(55,24) позволяют получать числа, удовлетворительные для большинства алгоритмов, требовательных к качеству случайных чисел. Значения  (a,b)=(97,33) позволяют получать очень качественные случайные числа и используются в алгоритмах, работающих со случайными векторами высокой размерности. Описанный фибоначчиев генератор случайных чисел (с лагами 20 и 5) используется в широко известной системе [Matlab](https://ru.wikipedia.org/wiki/Matlab" \o "Matlab) (автором первой версии этой системы был Д. Каханер).

[Брюс Шнайер](https://ru.wikipedia.org/wiki/%D0%A8%D0%BD%D0%B0%D0%B9%D0%B5%D1%80,_%D0%91%D1%80%D1%8E%D1%81) в своей работе «Прикладная криптография» приводит примеры 3-х алгоритмов на основе аддитивных генераторов.

Среди них алгоритм Fish, Pike и Mush.

Fish — это аддитивный генератор, основанный на методах, используемых в прореживаемом генераторе. Он выдает поток 32-битовых слов, которые могут быть использованы (с помощью XOR) с потоком открытого текста для получения шифротекста или с потоком шифротекста для получения открытого текста. Название алгоритма представляет собой сокращение от Fibonacci shrinking generator — прореживаемый генератор Фиббоначи.

Во первых используйте два следующих аддитивных генератора. Ключом является начальные состояния этих генераторов.

Эти последовательности прореживаются попарно в зависимости от младшего значащего бита Bi. Если его значение равно 1, то пара используется, если 0 – игнорируется. Cj – последовательность используемых слов Ai, a Dj – это последовательность используемых слов Bi. Для генерации двух 32-битовых слов-результатов K2j и K2j+1 эти слова используются парами – C2j, C2j+1, D2j, D2j+1.

Этот алгоритм быстр, на процессоре реализация Fish на языке С шифрует данные со скоростью 15-Мбит/с. К сожалению он также не безопасен, порядок вскрытия составляет около 240.

[Pike](https://ru.wikipedia.org/wiki/PIKE) — это обедненная и урезанная версия Fish, предложенная [Россом Андерсоном](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B4%D0%B5%D1%80%D1%81%D0%BE%D0%BD,_%D0%A0%D0%BE%D1%81%D1%81), тем, кто взломал Fish. Он использует три аддитивных генератора. Например:

Для генерации слова потока ключей взгляните на биты переноса при сложении. Если все три одинаковы (все нули или все единицы), то тактируются все три генератора. Если нет, то тактируются только два совпадающих генератора. Сохраните биты переноса для следующего раза. Окончательным выходом является XOR выходов трех генераторов.

Pike быстрее Fish, так как в среднем для получения результата нужно 2.75 действия, а не 3.

Mush представляет собой взаимно прореживающий генератор. Его работу объяснить легко. Возьмем два аддитивных генератора: А и В. Если бит переноса А установлен, тактируется В. Если бит переноса В установлен, тактируется А. Тактируем А и при переполнении устанавливаем бит переноса. Тактируем В и при переполнении устанавливаем бит переноса. Окончательным выходом является XOR выходов А и В. Проще всего использовать те же генераторы, что и в Fish:

В среднем для генерации одного выходного слова нужно три итерации генератора. И если коэффициенты аддитивного генератора выбраны правильно и являются взаимно простыми, длина выходной последовательности будет максимальна.

Таким образом, с учетом всех достоинств и недостатков существующего алгоритма был выбран алгоритм Mush с целью получения выходной последовательности максимальной длины.

**4.4** Описание алгоритма получения последовательности с нужным распределением

В программном модуле TestSystem для получения случайной последовательности тестовых данных был выбран алгоритм Mush.

Данный алгоритм позволяет генерировать последовательность с равномерным распределением.

Алгоритм был протестирован с помощью разработанного для этой цели программного средства. В качестве результата проверки алгоритма выведена гистограмма частоты попадания значений в 20 интервалов.

Была сгенерирована выбрка из 10 000 значений.

Результаты показаны на рисунке 4.2.

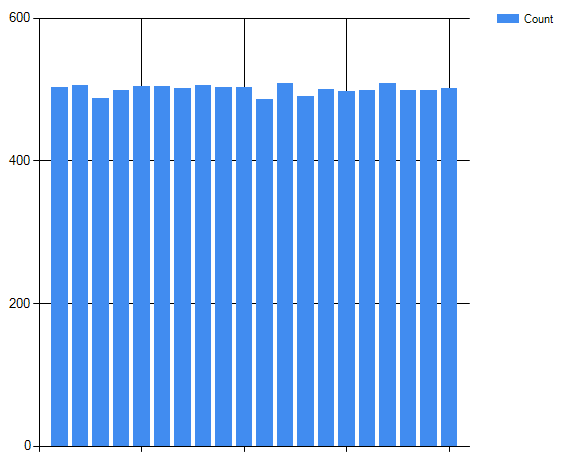


Рисунок 4.2 – Гистограмма алгоритма Mush

Далее алгоритм Mush был модифицирован с целью имитации Гауссовского распределения.

Гистограмма показана на рисунке 4.3.

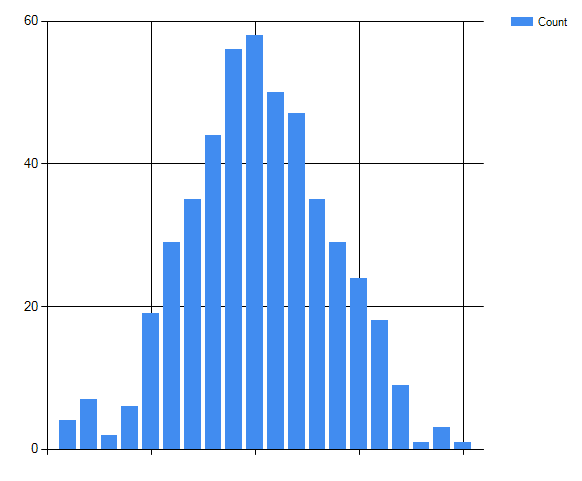


Рисунок 4.2 – Гистограмма модифицированного алгоритма Mush. Имитация гауссовского распределения.

**7** ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ WEB-СИСТЕМЫ ДЛЯ ТЕСТИРОВАНИЯ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ

**7.1** Введение

Целью дипломного проекта является разработка веб-системы для тестирования и оценки алгоритмов машинного обучения. Алгоритмы машинного обучения имеют широкое распространение в современном обществе. Они применяются для распознавания речи, распознавания рукописного текста, технической диагностики, медицинской диагностики, финансового прогнозирования и других сферах. Основная их задача – выполнять задачу распознавания на основе обучения на тестовых данных.

В настоящее время любой разработчик может предложить свой вариант алгоритма, который будет эффективен для решения определённой задачи. В связи с этим возникает необходимость систематизации, оценки алгоритмов, для чего и будет использоваться данное ПО.

В данном разделе приводится экономическое обоснование описанной разработки.

**7.2** Расчет объема ПО

Исходные данные для расчетов представлен в таблице 7.1. Перечень функций, используемых в программном продукте представлен в таблице 7.2. Объем функций по каталогу составлен на основе приложения 2 методического пособия.

Таблица 7.1 – Исходные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование показателей | Букв. обозн. | Ед.  измер. | Кол-во |
| 1 | 2 | 3 | 4 |
| Группа сложности | - | единиц | 1 |
| Коэффициент новизны | Кн | единиц | 1 |
| Поправочный коэффициент, учитывающий использование типовых программ | Кт | единиц | 0,7 |
| Дополнительный коэффициент сложности | Ксл | единиц | 0,26 |
| Тарифная ставка 1-го разряда | Тм1 | тыс. руб. | 1460 |

*Продолжение таблицы 7.1*

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Установленная плановая продолжительность разработки | Тр | год | 0,25 |
| Продолжительность рабочего дня | Тч | часов | 8 |
| Установленный фонд рабочего времени | Фрв | часов | 236 |
| Норматив дополнительной заработной платы | Нд | % | 20 |
| Норматив расходов на командировки | Нрнк | % | 10 |

Общий объемПО рассчитывается по формуле:

 (7.1)

где *V0* – общий объем ПО, строк исходного кода (LOC);

*Vi* – объем *i*-ой функции ПО, строк исходного кода (LOC).

Так как разработка проекта ведется на языке C#, объем кода на котором как правило занимает меньше строк, чем на C++, можно использовать уточненный объем *Vy* = 7000 LOC вместо 5060 [31].

Таблица 7.2- Перечень функций ПО и их объемы

|  |  |  |  |
| --- | --- | --- | --- |
| № функции | Наименование (содержание) функции | Объем функции (LOC) | |
| по каталогу Vi | уточненный Vyi |
| 1 | 2 | 3 | 4 |
| 101 | Организация ввода информации | 150 | 100 |
| 102 | Контроль, предварительная обработка и ввод информации | 450 | 400 |
| 109 | Организация ввода/вывода информации в интерактивном режиме | 320 | 450 |
| 204 | Обработка наборов и записей  базы данных | 2670 | 2000 |
| 301 | Формирование последователь-  ного файла | 290 | 400 |

*Продолжение таблицы 7.2*

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 305 | Обработка файлов | 720 | 600 |
| 309 | Формирование файла | 1020 | 300 |
| 506 | Обработка ошибочных и сбой-  ных ситуаций | 410 | 410 |
| 507 | Обеспечение интерфейса между  компонентами | 970 | 400 |
|  | Итого | 7000 | 5060 |

**7.3** Расчет нормативной трудоемкости, заработной платы исполнителей

Нормативная трудоемкость ПО(*Тн*) определяется по таблице укрупненных норм рабочего времени на разработку ПО. Объему 5060 условных машинных команд для 1-й группы сложности ПО соответствует нормативная трудоемкость *Тн* = 170 человеко-дней.

Общая трудоемкость небольших проектов рассчитывается по формуле:

 (7.2)

где *Кс* – коэффициент, учитывающий сложность ПО;

*Кн* – коэффициент, учитывающий степень новизны ПО, *Кн* = 1;

*Кт* – поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей, *Кт* =0,7.

Коэффициент, учитывающий сложность ПО рассчитывается по следующей формуле:

 (7.3)

где *Кi* – коэффициент, соответствующий степени повышения сложности ПО за счет конкретной характеристики.

*Кс* = 1 + 0,06 + 0,07+0,26 = 1,39

*То* = 170· 1,39· 1 · 0,7 = 165 чел/дн

Численность исполнителей проекта с учетом срока разработки и эффективного фонда рабочего времени работника рассчитывается по формуле:

 (7.4)

где *Тр* – срок разработки проекта (лет), *Тр* = 3 месяцев = 0,25 года;

*Фэф*– эффективный фонд времени работы одного работника в течении года (дн.).

 (7.5)

где *Дг* – количество дней в году;

*Дп* – количество праздничных дней в году;

*Дв* – количество выходных дней в году;

*До* – количество дней отпуска в году.

*Фэф*= 236 дней

*Чр*= 165/ (0,25· 236) =3 человека

Далее рассчитывается для каждого исполнителя месячная тарифная ставка по формуле:

 (7.6)

где *Тм* – месячная тарифная ставка;

*Тм1* – месячная тарифная ставка 1-го разряда;

*Тк* – тарифный коэффициент, соответствующий установленному тарифному разряду.

Разработкой будут производить три специалиста с высшим образованием, тарифный коэффициент – 2,48.

*Тм1* = 1 460 000 · 2,48 = 3 620 800 руб.

*Тм2* = 1 460 000 · 5 · 2,48 = 3 620 800 руб.

*Тм3* = 1 460 000 · 5 · 2,48 = 3 620 800 руб.

Кроме того, рассчитается также часовая тарифная ставка:

 (7.7)

где *Тч* – часовая тарифная ставка (руб.);

*Тм* – месячная тарифная ставка (руб.);

*Фрв* – установленный фонд рабочего времени, *Фрв* = 170 (ч).

*Тч1* = 3 620 800 / 170 = 21300 руб.

*Тч2* = 3 620 800 / 170 = 21300 руб.

*Тч3* = 3 620 800 / 170 = 21300 руб.

Основную заработную плату исполнителей рассчитывается по формуле:

 (7.8)

где *Зоi* – основная зарплата исполнителей (руб.);

*Тчi* – часовая тарифная ставка i-го исполнителя (руб.);

*Тч* – количество часов работы в день (ч);

*Фэi* – эффективный фонд рабочего времени i-го исполнителя (дней);

*К* – коэффициент премий. Равен 1,5.

*Зоi* = 3· (21300· 8 · 79· 1,5) = 60 573 854(руб.).

Дополнительная заработная плата определяется по нормативу в процентах к основной заработной плате:

 (7.9)

где *Здi* – дополнительная заработная плата исполнителей на конкретное ПО (руб.);

*Нд*– норматив дополнительной заработной платы в целом по научной организации, *Нд*=20%.

*Здi*= 60 573 854· 20 / 100 = 12 114 770(руб.).

Для дальнейших расчетов определяется сумма отчислений в фонд социальной защиты населения (*Зсз*) по формуле:

 (7.10)

где *Нсз*− норматив отчислений в фонд социальной защиты, *Нсз*=34%.

*Зсз*= (60 573 854 + 12 114 770) · 34 / 100= 24 714 132(руб.).

Отчисления по обязательному страхованию населения:

 (7.11)

где *Нстр*− норматив отчислений по обязательному страхованию, *Нстр* = 0,6%.

*Зстр*= (60 573 854 + 12 114 770) ·0,6/ 100 = 436 131(руб.).

Расходы по статье «Материалы» (*Мi*) определяются на основании сметы затрат на разработку ПО с учетом действующих нормативов (магнитные носители, бумага, красящие ленты, картриджи и другие материалы, необходимые для разработки ПО):

(7.12)

где *Нмз* – норма расхода материалов на 100 строк исходного кода ПО (руб.);

– общий объем ПО (строк исходного кода).

Тогда получается:

*Мi*= 60 573 854 \* 3 / 100 = 1 817 215(руб.).

Расходы по статье «Спецоборудование» (*Рci*).

Для разработки программного обеспечения использовалась покупка специального доступа к удаленной виртуальной машине. Цена подписки составила *Pci*= 450 000 (руб.).

Расходы по статье «Машинное время» – оплата машинного времени, необходимого для разработки и отладки ПО – определяются по формуле:

 (7.13)

где *Цмi* – цена одного машино-часа (руб.), *Цмi*= 10000 (руб.);

*Hмв* – норматив расхода машинного времени на отладку 100 машинных команд (машино-часов), *Hмв*=15 ч/(100 команд).

*Рмi*= 10000 · 5060· 15· / 100 = 7 590 000 (руб.).

Расходы по статье «Научные командировки» определяются по нормативу в целом по организации, в процентах к основной заработной плате:

 (7.14)

где *Hрнк* – норматив расходов на командировки в целом по организации (%), *Нрнк*=30%. Тогда получается:

*Рнкi*= 60 573 854 · 30/ 100 = 18 172 156 (руб.).

Прочие затраты определяются по формуле:

 (7.15)

где *Нпз* – норматив прочих затрат по научной организации, *Нпз* = 20%.

*Пз*= 60 573 854 · 20 / 100 = 12 114 770 (руб.).

Накладные расходы (*Рнi*) вычисляются по формуле:

 (7.16)

где *Нрн* – норматив накладных расходов в целом по организации, *Нрн*=100%.

*Рн*= 60 573 854 · 100 / 100 = 60 573 854 (руб.).

Общая сумма расходов (*Спi*) определяются по формуле:

 (7.17)

Тогда получаем общую сумму расходов с учетом ранее подсчитанных накладных расходов, прочих затрат, расходов на научные командировки и других затрат:

*Спi* = 60 573 854 + 12 114 770 + 24 714 132 + 436 131 + 1 817 215+

+ 450 000 + 7 590 000 + 18 172 156 + 12 114 770 + 60 573 854 =

= 198 556 886 (руб.).

Далее рассчитывается прогнозируемая прибыль по создаваемому ПО:

 (7.18)

где *Пoi* – прибыль от реализации ПО заказчику (руб.);

*Урп* – уровень рентабельности ПО (%), *Урп*= 40%. Тогда:

*Пoi*= 198 556 886 · 40 / 100 = 79 422 754 (руб.).

Прогнозируемая цена без налогов (*Цр*) рассчитывается по формуле:

 (7.19)

*Цпi* = 198 556 886 + 79 422 754 = 277 979 640(руб.).

Налог на добавленную стоимость не учитывается, так как организация является резидентом парка высоких технологий и имеет льготы.

Прогнозируемая отпускная цена рассчитывается по формуле:

 (7.20)

*Цоi* = 277 979 640 (руб.).

Затраты на освоение ПО определяются по формуле:

 (7.21)

где *Но*– норматив расходов на освоение, *Но*= 10%.

*Роi*= 198 556 886 · 10 / 100 = 19 855 689 (руб.).

Затраты на сопровождения ПО (*РСИ*) также рассчитывается по формуле:

 (7.22)

где *Нс*– норматив расходов на сопровождение, *Нс*= 20%.

*Рсi*= 198 556 886 · 20 / 100 = 39 711 377 (руб.).

Прибыль, остающаяся в распоряжении организации-разработчика, равна *Пoi*. Налог на добавочную стоимость не учитывается, так как компания является резидентом парка высоких технологий.

*Пч* = 79 422 754 (руб.).

7.4 Расчет экономического эффекта от применения программного средства у пользователя

Исходные данные для определения экономического эффекта приведены в таблице 7.3.

Общие капитальные затраты для пользователя составят:

 (7.23)

где *Кпр* - затраты пользователя на приобретение ПО по отпускной цене у разработчика с учетом стоимости услуг по эксплуатации (руб.);

*Кос* - затраты пользователя на освоение ПО (руб.);

*Кс* – затраты пользователя на оплату услуг по сопровождению ПО (руб.);

*Коб* - затраты на пополнение оборотных средств в связи с использованием нового ПО (руб.).

*Ко* = 277979640 + 19855688+ 39711377 + 880000 = 338 426 706 (руб.).

Экономия затрат на заработную плату:

 (7.24)

где *Сзе* - экономия затрат на заработную плату при решении задач c использованием нового ПО (руб.);

*А2*–количество типовых задач, решаемых за год (задач).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 7.3 – Исходные данные для определения экономического эффекта | | | | |
| Наименование показателей | Обозна-чение | Ед. измер. | Значение показателя | |
| в базовом варианте | в новом варианте |
| 1 | 2 | 3 | 4 | 5 |
| Капитальные вложения, включая затраты пользователя на приобретение ПО | Кпр | руб. | - | 277 979 640 |
| Затраты на освоение ПО | Кос | руб. | - | 19 855 688 |
| Затраты на сопровождение ПО | Кс | руб. | - | 39 711 377 |
| Затраты на пополнение оборотных средств в связи с эксплуатацией нового ПО | Коб | руб. | - | 880 000 |
| Коэффициент начислений на зарплату | Кнз |  | 1,5 | 1,5 |
| Количество типовых задач, решаемых за год | Зт | задача | 1 900 | 1 900 |
| Средняя трудоемкость работ | Тс1, Тс2 | чел.-час на задачу | 6.5 | 1.5 |
| Цена 1-го часа простоя системы | Цм | руб. | 10 000 | 10 000 |

Экономия затрат на заработную плату в расчете на 1 задачу рассчитывается по формуле:

 , (7.25)

где *Зсм* - среднемесячная заработная плата одного программиста (руб.);

*Тс1*, *Тс2* - трудоемкости работ (человеко-часов)в расчете на 1 задачу;

*Тч* - количество часов работы в день (ч);

*Др* - среднемесячное количество рабочих дней.

*Сзе* = 3 620 800∙ (6.5 – 1.5) : 8 : 21,5 = 105 255 (руб.).

*Сз* = 105 255 ∙ 1 900 = 199 986 046 (руб.).

Экономия затрат за счет сокращения начислений на заработную плату (*Соз*) при коэффициенте начислений равном 1,5:

*Сн*= 199 986 046 ∙ 1,5= 299 979 069 (руб.).

Экономия затрат на простой системы (*Сс*):

*Сс*=(*П*-*П2*) \* *Сп* \* *Дрг* / 60 (7.26)

*Сс* = (40 - 10) ∙ 10 000 \* 225 = 1 125 000 (руб.).

Общая годовая экономия текущих затрат вычисляется по формуле:

*Со=Сн+Сс* (7.27)

*Со* = 299 979 069 + 1 125 000 = 301 104 069 (руб.)

Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль – дополнительная прибыль, остающаяся в его распоряжении, которая определяется по формуле:

, (7.28)

где *Нп* – ставка налога на прибыль.

*ΔПч* = 301 104 069 – (301 104 069 ∙ 18) / 100= 246 905 337(руб.)

В процессе использования нового ПО чистая прибыль в конечном итоге возмещает капитальные затраты. Однако полученные при этом суммы результатов (прибыли) и затрат (капиталовложений) по годам приводят к единому времени – расчетному году (за расчетный год принят 2015 год) путем умножения результатов и затрат за каждый год на коэффициент приведенияαt, который рассчитывается по формуле:

, (7.29)

где *Ен* – норматив приведения разновременных затрат и результатов, *Ен*=15%;

*t* – номер года, результаты и затраты которого приводятся к расчетному (2015 - 1, 2016 - 2, 2017 - 3 и так как);

*tр*– расчетный год (2015).

Следовательно, при решении данной задачи коэффициентам приведения (*αt*) по годам будут соответствовать следующие значения:

*αt0* = 1,000 – первый расчетный год;

*αt1* = (1 + 0,15)2015-2016= 0,869 – 2016 расчетный год;

*αt2* = (1 + 0,15)2016-2017= 0,756 – 2017 расчетный год;

*αt3* = (1 + 0,15)2017-2018= 0,658 – 2018 расчетный год.

Сведем данные расчета экономического эффекта в таблицу 7.4.

Таблица 7.4 – Расчет экономического эффекта от использования нового ПО

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Показатели | Ед. изм. | Годы | | | |
| 2014 | 2015 | 2016 | 2017 |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Результаты: | | | | | |
| Прирост прибыли за счет экономии затрат (Пч) | тыс. руб. |  | 246905 | 246905 | 246905 |
| То же с учетом фактора времени | руб. |  | 214560 | 186660 | 162463 |
| Затраты: | | | | | |
| Приобретение ПО (Кпр) | тыс. руб. | 277 980 |  |  |  |
| Освоение ПО (Кос) | тыс. руб. | 19 856 |  |  |  |
| Сопровождение (Кс) | тыс. руб. | 39 711 |  |  |  |
| Пополнение оборотных средств (Коб) | тыс. руб. | 880 |  |  |  |
| Всего затрат | тыс. руб. | 338427 |  |  |  |
| То же с учетом фактора времени | тыс. руб. | 338427 |  |  |  |

*Продолжение таблицы 7.4*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Экономический эффект: |  |  |  |  |  |
| Превышение результатов над затратами | тыс. руб. | -338427 | 214560 | 186660 | 162463 |
| То же нарастающим итогом | тыс. руб. | -338427 | -123867 | 62793 | 225256 |
| Коэффициент приведения | ед. | 1 | 0,869 | 0,756 | 0,658 |

Из расчетов видно, что веб-система для оценки алгоритмов машинного обучения окупается в пределах двух лет.

**7.5** Вывод

В результате разработки описанного программного обеспечения, организация получит 79 422 754 рублей прибыли. Пользователь получит прибыль в результате сокращения затрат на выполнение работ и уменьшения простоя системы. Затраты на покупку разрабатываемого ПО окупаются в пределах двух лет и, следовательно, разработка является экономически целесообразной.

К 2017 году пользователь ПО сможет сэкономить 225 256 000 рублей в результате использования разрабатываемой системы. Кроме того, благодаря использованию расширяемой архитектуры, веб-система сможет развиваться и выходить как обновленный продукт, принося в будущем большую прибыль предприятию.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Системы тестирования алгоритмов классификации: MLcomp, TunedIt и Полигон [Электронный ресурс]. – Режим доступа: http://www.machinelearning.ru/wiki/images/1/1b/Lisitsa10iip.pdf.

[2] Полигон алгоритмов / Документация [Электронный ресурс]. – Режим доступа: http://www.machinelearning.ru/wiki/index.php?title= Полигон.

[3] Cross validation [Электронный ресурс]. – Режим доступа : http://www.cs.cmu.edu/~schneide/tut5/node42.html.

[4] TunedIt. Research documentation [Электронный ресурс]. – Режим доступа : http://tunedit.org/research/docs.

[5] Weka – ARFF (stable version) [Электронный ресурс]. – Режим доступа : http://weka.wikispaces.com/ARFF+%28stable+version%29.

[6] Object Relationship Mapping (ORM) [Электронный ресурс]. – Режим доступа : http://tech.lalitbhatt.net/2014/07/object-relationship-mapping-orm.html.

[7] MSDN. Layered Application [Электронный ресурс]. – Режим доступа : https://msdn.microsoft.com/en-us/library/ff650258.aspx.