Содержание

**Обозначения и сокращения**

ПО – программное обеспечение

АРМ – автоматизированное рабочее место

БД – база данных

ИС – информационная система

ОС – операционная система

ПК – персональный компьютер СУБД – системы управления базами данных

UML – унифицированный язык моделирования

**Введение**

В наше время информационные технологии используются почти во всех сферах деятельности человека, в связи с этим возникает потребность производить большое количество программ и решать более сложные задачи, в связи с этим необходимо тестировать каждый программный продукт. Для всего этого зачастую используют тестирование.

Тестирование программного продукта – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом

Тестирование является одним из наиболее устоявшихся способов обеспечения качества разработки программного обеспечения и входит в набор эффективных средств современной системы обеспечения качества программного продукта.

В процессе обучения преподавателями, которые ведут дисциплины по программированию, приходится очень много времени тратить на проверку решения задач студентами. Необходимо в очень ограниченное время проверить на соответствие разработанной программы с условием задачи, а также есть ли проверка на граничные условия в программе. Поэтому разработка программы онлайн-тестирования решения задач по программированию является актуальной задачей.

Целью бакалаврской работы является разработка информационной системы тестирования программ на языке C++.

Для достижения поставленной цели бакалаврской работы необходимо решить ряд следующих задач:

– изучить имеющиеся проблемы тестирование программ;

– проанализировать и выбрать программное обеспечение и инструменты, необходимые для разработки системы;

– сформулировать технические требования и рекомендации к необходимой программе тестирования;

– выполнить UML-моделирование;

– спроектировать базу данных;

– выбрать инструментальные средства для разработки ИС;

– описать инструкцию пользователя по использованию разработанной информационной системы тестирования

# 1 Анализ технологии программирования в сфере тестирования программ. Постановка задачи

## 1.1 Анализ технологии тестирования , описание стандартов

В наше время информационные технологии используются почти во всех сферах деятельности человека, в связи с этим возникает потребность производить большое количество программ и решать более сложные задачи, в связи с этим необходимо тестировать каждый программный продукт. Для всего этого зачастую используют тестирование.

Тестирование программного продукта – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом

Тестирование является одним из наиболее устоявшихся способов обеспечения качества разработки программного обеспечения и входит в набор эффективных средств современной системы обеспечения качества программного продукта. Качество программного продукта характеризуется набором свойств, определяющих, насколько продукт "хорош" с точки зрения заинтересованных сторон, таких как заказчик продукта, спонсор, конечный пользователь, разработчики и тестировщики продукта, инженеры поддержки, сотрудники отделов маркетинга, обучения и продаж. Каждый из участников может иметь различное представление о продукте и о том, насколько он хорош или плох, то есть о том, насколько высоко качество продукта. Таким образом, постановка задачи обеспечения качества продукта выливается в задачу определения заинтересованных лиц, их критериев качества и затем нахождения оптимального решения, удовлетворяющего этим критериям.

Тестирование программного продукта с точки зрения классификации по целям должно делиться на два класса:

- Функциональное тестирование;
- Нефункциональное тестирование.

Под функциональным тестированием понимается проверка соответствия программного продукта функциональным требованиям, указанным в техническом задании на создание это продукта. При функциональном тестировании проверяется выполняет ли программный продукт все функции, которые должен.

Для проведения функционального тестирования персоналом отдела технического контроля разрабатывается документ программа и методика испытаний функционала приложения (ПМИ). Документ ПМИ содержит перечень сценариев тестирования программного продукта с подробным описанием шагов. Каждый шаг сценария тестирования характеризуется действиями пользователя (специалиста по тестированию) и ожидаемыми результатами – ответной реакции программы на эти действия. Программа и методика испытаний обязана имитировать эксплуатацию программного продукта в реальном режиме. Это означает, что сценарий тестирования должен быть построен на основе анализа операций, которые будут выполнять будущие пользователи системы, а не быть искусственно составленной последовательностью понятных только разработчику манипуляций.

Обычно, функциональное тестирование проводится на двух уровнях:

– Компонентное (модульное) тестирование. Тестирование отдельных компонентов программного продукта, сфокусированное на их специфике, назначении и функциональных особенностях.

– Интеграционное тестирование. Данный вид тестирования проводится после компонентного тестирования и направлен на выявление дефектов взаимодействия различных подсистем на уровне потоков управления и обмена данными.

Нефункциональное тестирование оценивает такие качества программного продукта, как например, эргономику или производительность. При нефункциональном тестирование проверяет соответствие программного продукта нефункциональным требования из технического задания. Как и в

случае с функциональным тестированием, для нефункционального разрабатывается программа и методика испытаний.

Главным ориентиром в определение качества тестирования является соответствие процедуры его проведения формальными правилами и стандартам относительно программного обеспечения.

Стандарты относящиеся к тестированию:

– IEEE 829-2008 IEEE Standard for Software and System Test Documentation;

– ANSI/IEEE 1008-1987 Standard for Software Unit Test;

– ISO/IEC/IEEE 29119-1:2013 Software and system engineering - Software testing – Part1: Concepts and definitions;

– ISO/IEC/IEEE 29119-2:2013 Software and system engineering - Software testing – Part2: Test processes;

– ISO/IEC/IEEE 29119-3:2013 Software and system engineering - Software testing – Part3: Test documentation.

### 1.2 Классификация видов и методов тестирования

Существует несколько признаков, по которым принято производить классификацию видов тестирования. Обычно выделяют следующие:

По объекту тестирования:

– Функциональное тестирование проверка соответствия программного продукта функциональным требованиям, указанным в техническом задании на создание это продукта. При функциональном тестировании проверяется выполняет ли программный продукт все функции, которые должен.

– Тестирование производительности проводится с целью определения, как быстро работает вычислительная система или её часть под определённой нагрузкой. Также может служить для проверки и подтверждения других атрибутов качества системы, таких как масштабируемость, надёжность и потребление ресурсов.

– Юзабилити-тестирование - исследование, выполняемое с целью определения, удобен ли некоторый искусственный для его предполагаемого применения. Таким образом, проверка эргономичности измеряет эргономичность объекта или системы. Проверка эргономичности сосредоточена на определённом объекте или небольшом наборе объектов, в то время как исследования взаимодействия человек-компьютер в целом — формулируют универсальные принципы.

– Тестирование безопасности - это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

– Тестирование совместимости - вид нефункционального тестирования, основной целью которого является проверка корректной работы продукта в определенном окружении.

По знанию внутреннего строения системы:

– Тестирование черного ящика - это стратегия или метод тестирования, базируется только лишь на тестировании по функциональной спецификации и требованиям, при этом не смотря во внутреннюю структуру кода и без доступа к базе данных. Фактически мы знаем какой должен быть результат при определенном наборе данных, которые подаются на вход. Результат проверяем на уровне простого пользователя. На данный момент такая стратегия является наиболее часто применима в IT компаниях.

– Тестирование белого ящика - тестирование кода на предмет логики работы программы и корректности её работы с точки зрения компилятора того языка, на котором она писалась. Тестирование по стратегии белого ящика, также называемое техникой тестирования, управляемой логикой программы, позволяет проверить внутреннюю структуру программы. Исходя из этой стратегии тестировщик получает тестовые данные путем анализа логики работы программы.

По степени автоматизации:

– Ручное тестирование - часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно проводится тестировщиками или обычными пользователи путем моделирования возможных сценариев действия пользователя. Задача тестировщика заключается в поиске наибольшего количества ошибок. Он должен хорошо знать наиболее часто допускаемые ошибки и уметь находить их за минимально короткий период времени. Остальные ошибки, которые не являются типовыми, обнаруживаются только тщательно созданными наборами тестов. Однако, из этого не следует, что для типовых ошибок не нужно составлять тесты.

– Автоматизированное тестирование - процесс тестирования программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, производятся автоматически с помощью инструментов для автоматизированного тестирования. В свою очередь, инструмент для автоматизированного тестирования - это программное обеспечение, посредством которого осуществляется создание, отладка, выполнение и анализ результатов прогона тест-скриптов (Test Scripts — это наборы инструкций для автоматической проверки определенной части программного обеспечения).

По степени изолированности :

– Модульное тестирование - процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

– Интеграционное тестирование - одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию. Интеграционное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определённые в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования.

– Системное тестирование - это тестирование программного обеспечения (ПО), выполняемое на полной, интегрированной системе, с целью проверки соответствия системы исходным требованиям. Системное тестирование относится к методам тестирования чёрного ящика, и, тем самым, не требует знаний о внутреннем устройстве системы.

По времени проведения тестирования:

– Альфа-тестирование – использование незавершенной версии продукта, в которой реализована не вся функциональность, запланированная для данной версии продукта, штатными разработчиками и тестерами с целью выявления ошибок в работе реализованных моделей и функций, для их дальнейшего устранения.

– Бета-тестирование - интенсивное использование почти готовой версии продукта с целью выявления ошибок в его работе для их последующего устранения перед окончательным выходом продукта.

### 1.3 Статическое и динамическое, регрессионное тестирование

Статическое тестирование – тип тестирования, который предполагает, что программный код во время тестирования не будет выполняться. При этом само тестирование может быть как ручным, так и автоматизированным.

Статическое тестирование начинается на ранних этапах жизненного цикла ПО и является, соответственно, частью процесса верификации. Для этого

типа тестирования в некоторых случаях даже не нужен компьютер – например, при проверке требований.

Большинство статических техник могут быть использованы для «тестирования» любых форм документации, включая вычитку кода, инспекцию проектной документации, функциональной спецификации и требований.

Даже статическое тестирование может быть автоматизировано – например, можно использовать автоматические средства проверки синтаксиса программного кода.

Виды статического тестирования:

– Вычитка исходного кода программы;

– Проверка требований.

Динамическое тестирование – тип тестирования, который предполагает запуск программного кода. Таким образом, анализируется поведение программы во время ее работы.

Для выполнения динамического тестирования необходимо чтобы тестируемый программный код был написан, скомпилирован и запущен. При этом, может выполняться проверка внешних параметров работы программы: загрузка процессора, использование памяти, время отклика и т.д. – то есть, ее производительность.

Динамическое тестирование является частью процесса валидации программного обеспечения.

Кроме того, динамическое тестирование может включать разные подвиды, каждый из которых зависит от:

– Доступа к коду (тестирование белым и черным ящиками);

– Уровня тестирования (модульное, интеграционное и системное тестирование);

– Сферы использования приложения (функциональное , нагрузочное и тестирование безопасности).

Регрессионное тестирование – тестирование функциональности программного обеспечения, после внесения изменений на фазе системного тестирования или сопровождения продукта. После очередного релиза, регрессионные тесты подтверждают, что изменения не повлияли на работоспособность остальной функциональности приложения. Регрессионное тестирование выполняться вручную или средствами автоматизации тестирования.

Для регрессионного тестирования используются тест кейсы, написанные на ранних стадиях разработки и тестирования. Это дает гарантию того, что изменения в новой версии приложения не повредили уже существующую функциональность. Рекомендуется проводить автоматизацию регрессионных тестов, для ускорения последующего процесса тестирования и обнаружения дефектов на ранних стадиях разработки программного обеспечения.

Задачи регрессионного тестирования

Основная задача регрессионного тестирования – проверка того, что исправление ошибки не коснулось существующей функциональности. Из-за частого выполнения одних и тех же наборов сценариев, рекомендуется использовать автоматизированные регрессионные тесты, что позволит сократить сроки тестирования.

– Проверка и утверждение исправления ошибок;

– Тестирование последствия исправлений, так как внесение исправлений могут внести ошибку в код который работал ранее;

– Уменьшение стоимости и сокращение времени выполнения тестов.

Проблема регрессионного тестирования состоит в выборе между полным и частичным повторным тестированием. Частичное тестирование контролирует только части проекта, связанные с измененными компонентами. Повторное выборочное тестирование системы или компонентов для проверки сделанных модификаций, не должны приводить к непредусмотренным эффектам. Задача состоит в том, чтобы определить критерии “масштабов” изменений, с достижением которых необходимо проводить регрессионные тесты.

Преимущества регрессионного тестирования

– Сокращение количества дефектов в системе к моменту релиза;

– Исключение деградации качества системы при росте функциональности;

– Уменьшение вероятности критических ошибок при эксплуатации .

### 1.4 Тестовые сценарии

На этапе контроля качества реализованной функциональности используется тестовая документация, в которой записаны стандарты и альтернативные сценарии с приложением, используемые при тестировании каждой новой версии приложения.

Тестовая документация состоит из тестовых сценариев, то есть специальным образом разработанного описания последовательности действий в системе и ожидание поведения. Тестовые сценарии используются для проведения различных видов ручного тестирования:

– Функционального тестирования;

– Приемочного тестирования;

– Нагрузочного тестирования;

– Исследовательского тестирования;

– Smoke-тестирования.

Для разработки тестовых сценариев и выполнения тестов используются системы управления тестированием, существенно повышающие производительность тестировщиков, а также обеспечивающие видимость уровня качества приложений среди всех участников проекта.

Тестовые сценарии неразрывно связаны с требованиями, изменения в которых должны своевременно отображаться в тестовой документации, что позволяет сделать система управления жизненным циклом разработки приложений, при помощи механизма трассировок.

При выполнение теста тестировщик отмечает результат прохождения одного шага или тестового сценария в целом, прикрепляет обнаруженные ошибки и другую вспомогательную информацию: скриншоты, дампы, логи.

Тестовые сценарии удобно объединять в тест-планы по назначению:

– Тестирование новой версии продукта;
– Тестирование развертывания;
– Тестирование удобства использования;
– Конфигурационное тестирование;
– Тестирование безопасности;

### 1.5 Анализ инструментов автоматизации тестирования

Методы разработки программного обеспечения не стоят на месте, то же можно сказать и об используемых для разработки ПО инструментах и технологиях. В первую очередь, такое совершенствование необходимо для того , чтобы продуктивность и качество создаваемого продукта увеличились, время, необходимое для разработки, сократилось, а заказчик остался доволен полученным результатом. Таким образом, можно смело заявить, что тестирование играет важную роль в разработке качественного продукта.

В области тестирования ПО инструменты автоматизации, безусловно, играют ключевую роль. Рассмотрим некоторые из них:

– Selenium – является самым популярным фреймворком с открытым исходным кодом, предназначенным для автоматизации тестирования веб-приложений. Так же его можно рассматривать в качестве родоначальника некоторых современных инструментов автоматизации тестирования с открытым исходным кодом, например: Katalon Studio, Watir, Protractor и Robot Framework.

– Katalon Studio - это эффективный инструмент для автоматизации процесса тестирования веб-приложений, мобильных приложений и веб-сервисов. Katalon Studio может быть интегрирован в CI/CD, он прекрасно

работает в связке с популярными инструментами во время тестирования ПО: qTest, JIRA, Jenkins и Git. Для него предусмотрена функция — Katalon Analytics, благодаря которой пользователи получают полное представление о процессе тестирования. Для этого предусмотрены специальные отчеты, которые выводятся на экран пользователей в виде метрики, диаграмм и графиков.

– Unified Functional Testing – это популярный коммерческий инструмент для функционального тестирования. Он предоставляет полный набор функций для тестирования API, веб-сервисов, а также для тестирования графического интерфейса десктопных, мобильных и веб-приложений на всех существующих платформах. Для данного инструмента предусмотрена расширенная функция распознавания объектов на основе изображений, многоразовые тестовые компоненты и документация по автоматическому тестированию. UFT использует Visual Basic Scripting Edition, который может пригодиться для записи информации о выполненном тестировании, а также для управления объектами. UFT интегрирован с Mercury Business Process Testing и Mercury Quality Center. Инструмент поддерживает CI с помощью интеграции с инструментами CI, такими как Jenkins.

### 1.6 Техническое задание

Полное наименование системы: «Онлайн-тестировщик для проверки решений задач по программированию на языке C++ ».

Краткое наименование:« Тестировщик».

Назначение системы« Тестировщик» является проверка программ на работоспособность и корректность выполнения поставленной задачи.

Основными целями создания «Тестирование» являются:

– значительное сокращение времени на проверку решений задач по программированию;

– значительное сокращение времени на обработку результатов тестирования;

– более эффективное использование рабочего времени.

Для решения поставленных целей система должна решать следующие задачи:

– ввод входных данных в объект тестирования;

– вывод результата тестирования на экран;

– построение протокола прохождения тестирования для отчетности.

Объектом автоматизации являются процессы тестирования программ.

Процессы тестирования программ включают в себя:

– произведение непосредственного тестирования;

– составления протокола прохождения тестирования;

В состав «Тестирощик» должны входить следующие подсистемы:

– подсистема прохождения тестирования;

– подсистема хранения данных;

– подсистема формирования отчетности.

Подсистема прохождения тестирования обеспечивает выбор объекта тестирования и выполнение теста.

Подсистема хранения данных обеспечивает хранение данных о прохождении теста для дальнейшего анализа, вывода результата и передачи информации подсистеме формирования отчетности.

Подсистема формирования отчетности предназначена для создания и формирования отчетов в виде удобном для вывода на печатающие устройства на основе данных «Тестирощик» и вывода подготовленных отчетных форм на печать.

Для эксплуатации «Тестирощик» определены следующие роли:

– Студент

– Преподаватель

Основные обязанности которые должен выполнять преподаватель:

– тестирование программы студента ;

– запись результатов тестирования в БД .

### 1.7 Анализ среды программирования и инструментов для разарботки

Для разработки системы тестирования требуется:

– ОС Windows 10;

– Microsoft Visual Studio – интегрированная среда разработки;

– Microsoft SQL Server – реляционная система управления базами данных.

Microsoft Visual Studio - линейка продуктов компании Microsoft включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов ( например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server).

Microsoft SQL Server – система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

### 1.8 Вывод по первому разделу

В первой главе был выполнен анализ технологий тестирования, описаны стандарты тестирования, приведена классификация видов и методов тестирования.

Рассмотрены популярные инструменты тестирования программ.

В результате анализа были определены подходы к созданию информационной системы.

Описано техническое задание для разработки онлайн-тестировщика для проверки решений задач по программированию.

## 2 Проектирование системы тестирования

### 2.1 Требования к функциональным характеристикам

Разрабатываемое программное приложение тестирования будет содержать две подсистемы:

1) Подсистема прохождения тестирования.

Подсистема должна осуществлять тестирование программ и записывать данные в базу данных.

2) Подсистема хранения данных.

Подсистема хранения данных должна осуществлять хранение оперативных данных системы, данных для формирования аналитических отчетов, документов системы, сформированных в процессе работы программы.

### 2.2 UML-моделирование проектируемого приложения

1 Диаграмма вариантов использования

Диаграмма вариантов использования – диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Прецедент – это возможность моделируемой системы( часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе.

Актер – представляет собой некую роль, которую пользователь играет по отношению к системе. Актеры действуют в рамках прецедентов. Один актер может выполнять несколько прецедентов; и наоборот, в соответствии с одним прецедентом могут действовать несколько актеров.

Составим таблицу, и заполним ее необходимыми прецедентами, в таблице 2.1.

Таблица 2.1 – Прецеденты ИС

| **Прецеденты** | **Актер** | **Система** |
|---|---|---|
| **Преподаватель** | | |
| «Тестирование программы преподавателем» | Преподаватель должен протестировать программу . | Открывает выбранную программу для тестирования. |
| << include >> «Вход в систему » | Вводит свой логин и пароль. | Система проверяет введенные данные. |
| <<extend >> «Просмотр результатов тестирования» | Может просмотреть результаты тестирования. | Система предоставит возможность просмотреть результаты тестирования. |

Продолжение таблицы 2.1

| **Прецеденты** | **Актер** | **Система** |
|---|---|---|
| **Студент** | | |
| «Тестирование программы студентом» | Студент может протестировать свою программу. | Открывает выбранную программу для тестирования. |
| <<extend>>« Просмотреть результата тестирования» | Сотрудник, после прохождения теста, может просмотреть с каким результатом выполнилась его программа. | Система выводит сообщение с результатом тестирования. |

Таким образом, диаграмма вариантов прецедентов показана на рисунке 2.1.

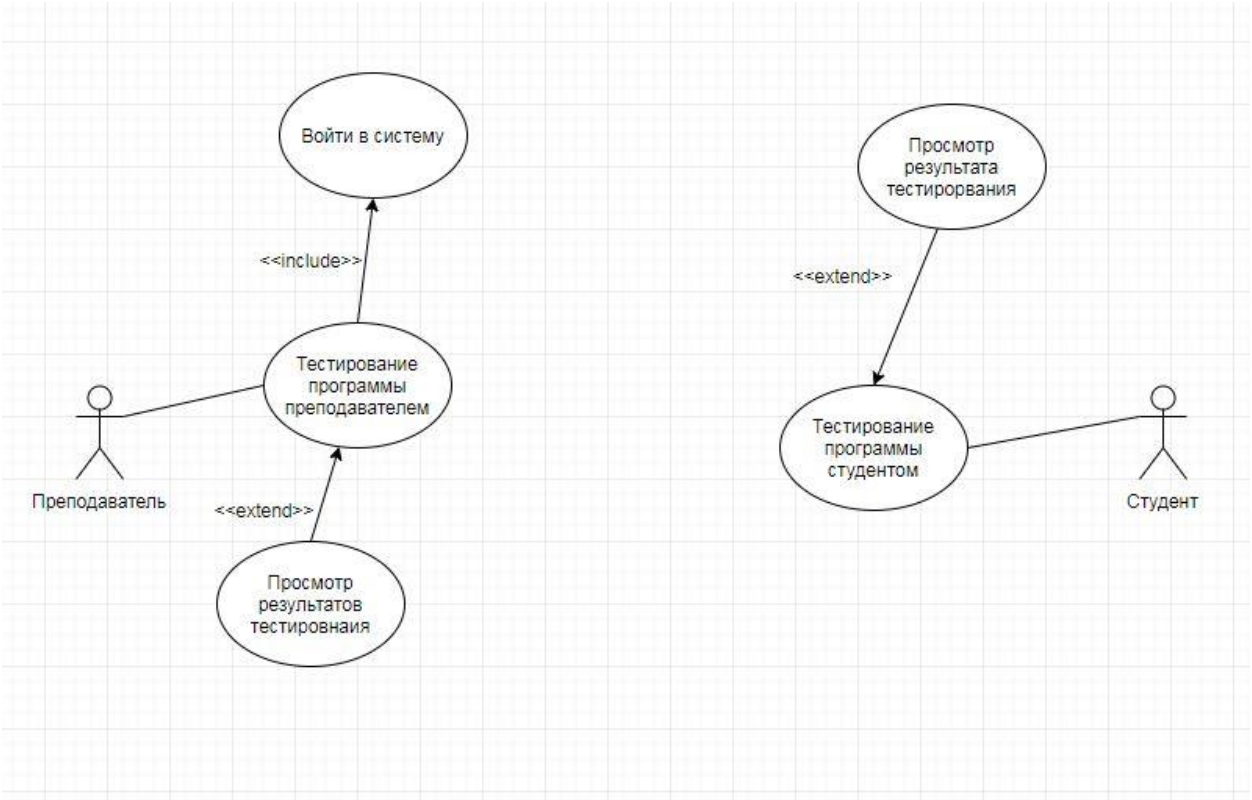



Рисунок 2.1 – Диаграмма вариантов использования

2 Диаграмма последовательности

Диаграмма последовательности( англ. sequence diagram) – диаграмма, на которой показано взаимодействие объектов (обмен между ними сигналами и сообщениями), упорядоченное по времени, с отражением продолжительности обработки и последовательности их проявления. Используется в языке UML. Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные« линии жизни» (англ. lifeline), отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

Диаграмма последовательности относится к диаграммам взаимодействия UML, описывающим поведенческие аспекты системы, но рассматривает взаимодействие объектов во времени.

На диаграмме, представленной на рисунке 2.2, показана последовательность действий прохождения тестирования преподавателем. Для начала преподавателю необходимо авторизоваться, для этого он вводит нужную информацию, система обратится к базе данных и вернет сообщение о успехе или неудаче. Далее можно проходить тестирование. Для этого выбирается нужная программа. В конце преподаватель может посмотреть результаты тестирования.

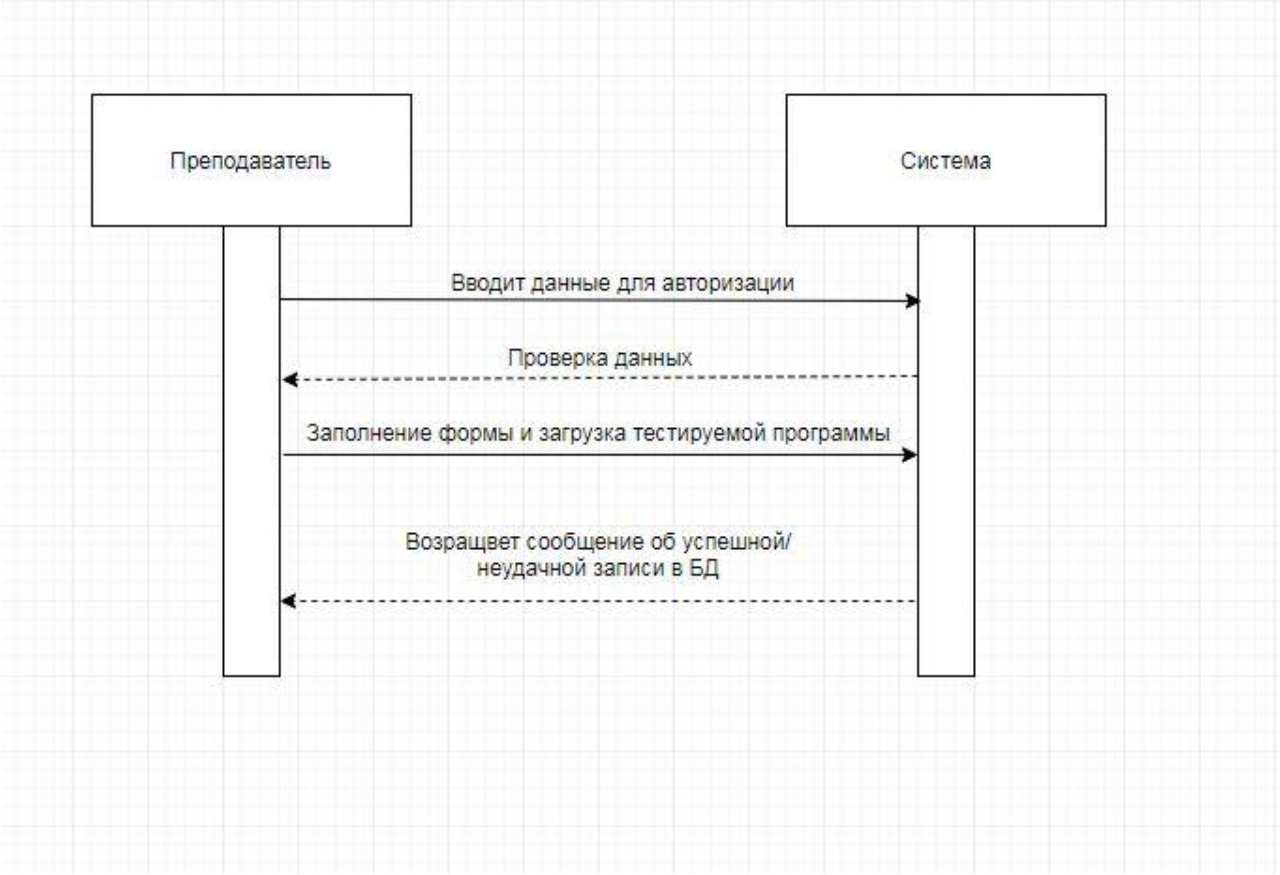



Рисунок 2.2 – Диаграмма последовательности «Тестирование программы преподавателем»

На диаграмме, представленной на рисунке 2.3, показана последовательность действий прохождения тестирования студентом. Для начала студент выбирает файл своей программы , затем система тестирует программу и выдает сообщение с результатом тестирования.

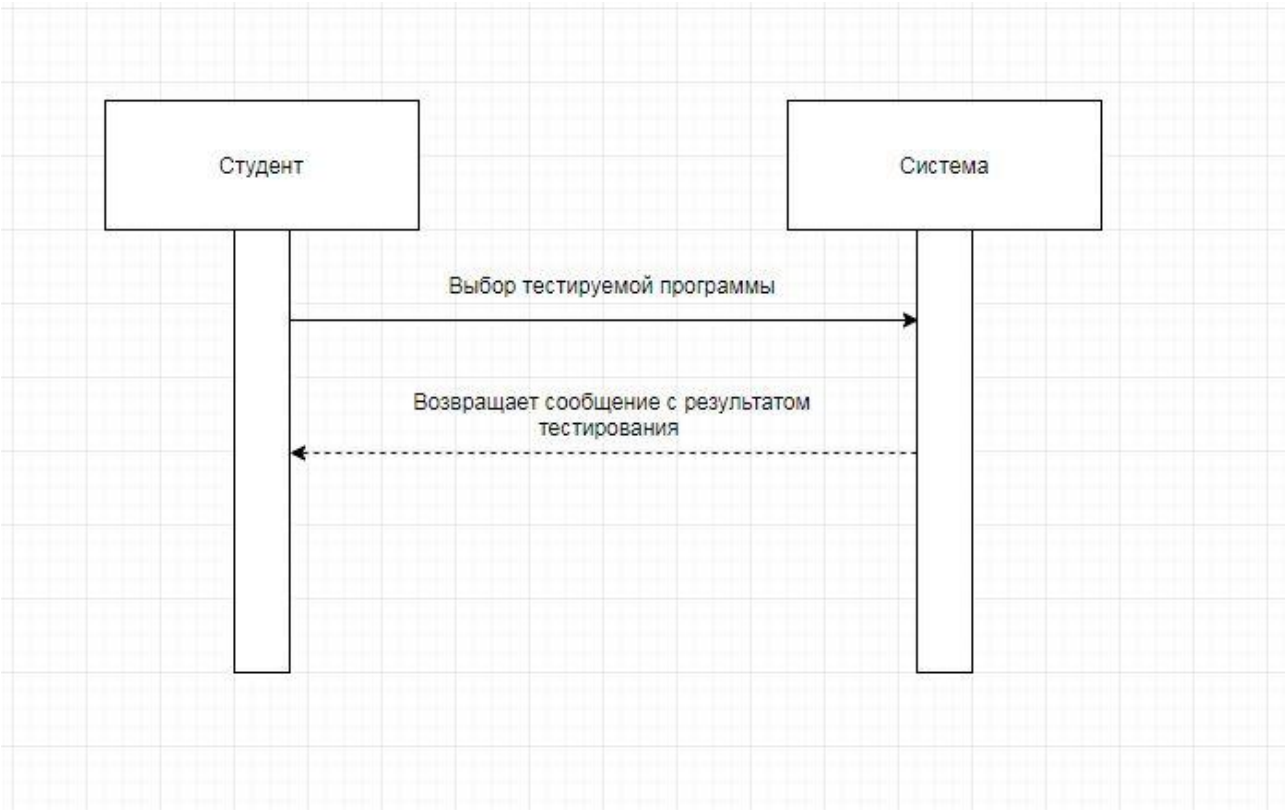



Рисунок 2.3 – Диаграмма последовательности «Тестирование программы студентом»

## 3 Диаграмма компонентов информационной системы

Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Диаграмма компонентов показана на рисунке 2.4.

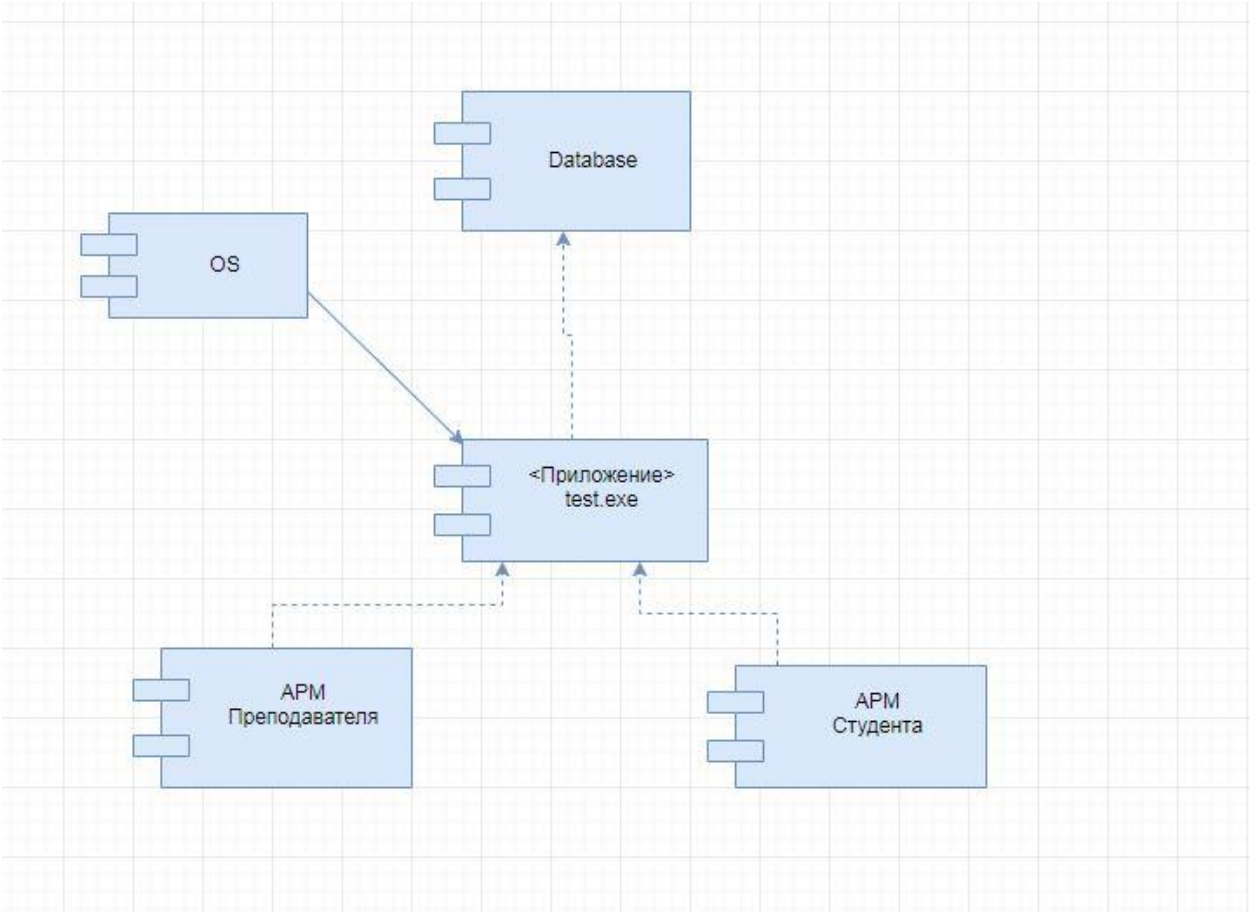



Рисунок 2.4 – Диаграмма компонентов информационной системы

4 Диаграмма деятельности

Диаграмма деятельности – диаграмма, на которой показано разложение некоторой деятельности на её составные части. Под деятельностью понимается спецификация исполняемого поведения в виде координированного последовательного и параллельного выполнения подчинённых элементов – вложенных видов деятельности и отдельных действий, соединённых между собой потоками, которые идут от выходов одного узла к входам другого.

Данная диаграмма деятельности для ИС «Тестировщик» показана на рисунках 2.5-2.6.

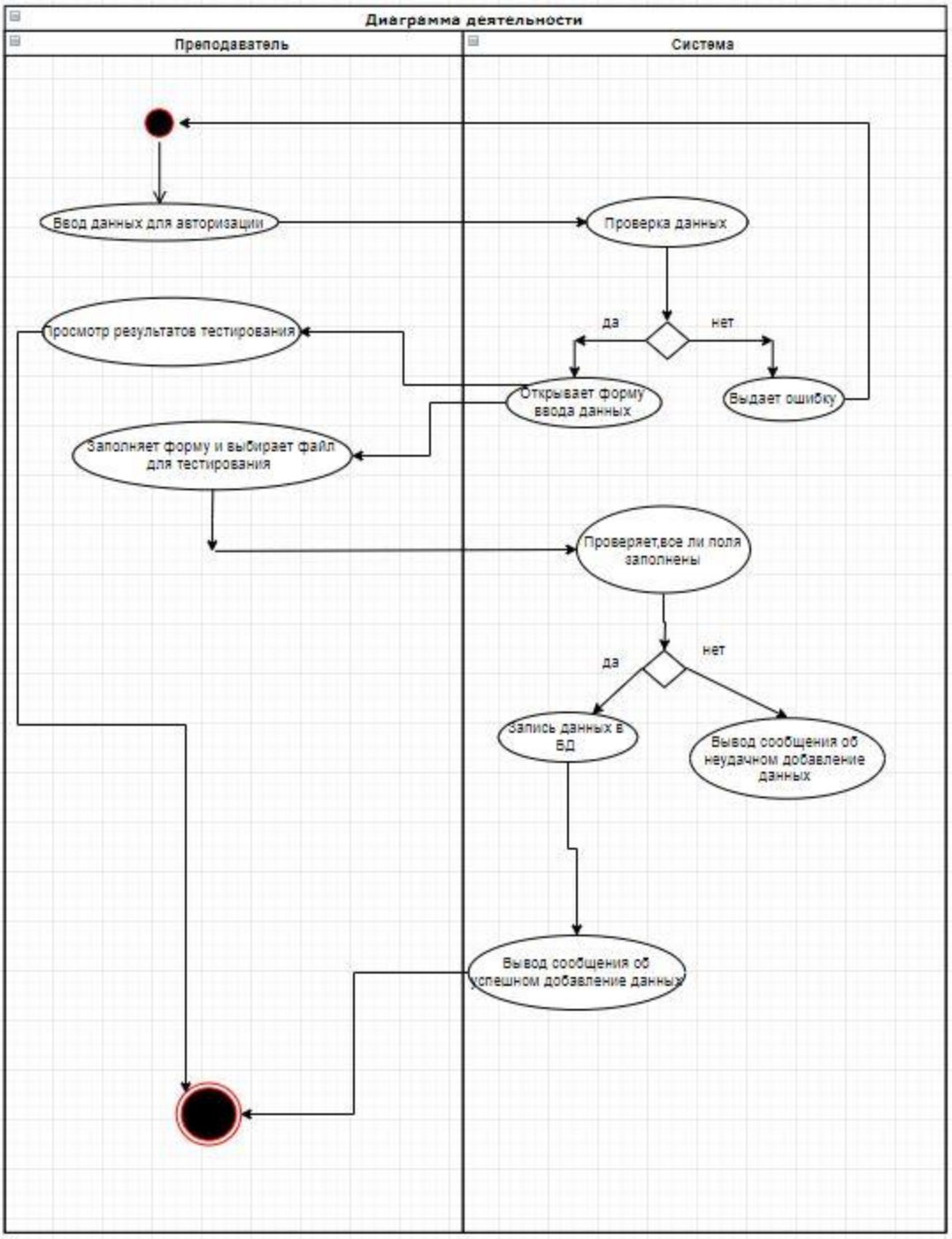



Рисунок 2.5 – Диаграмма деятельности« Прохождения тестирования преподавателем»

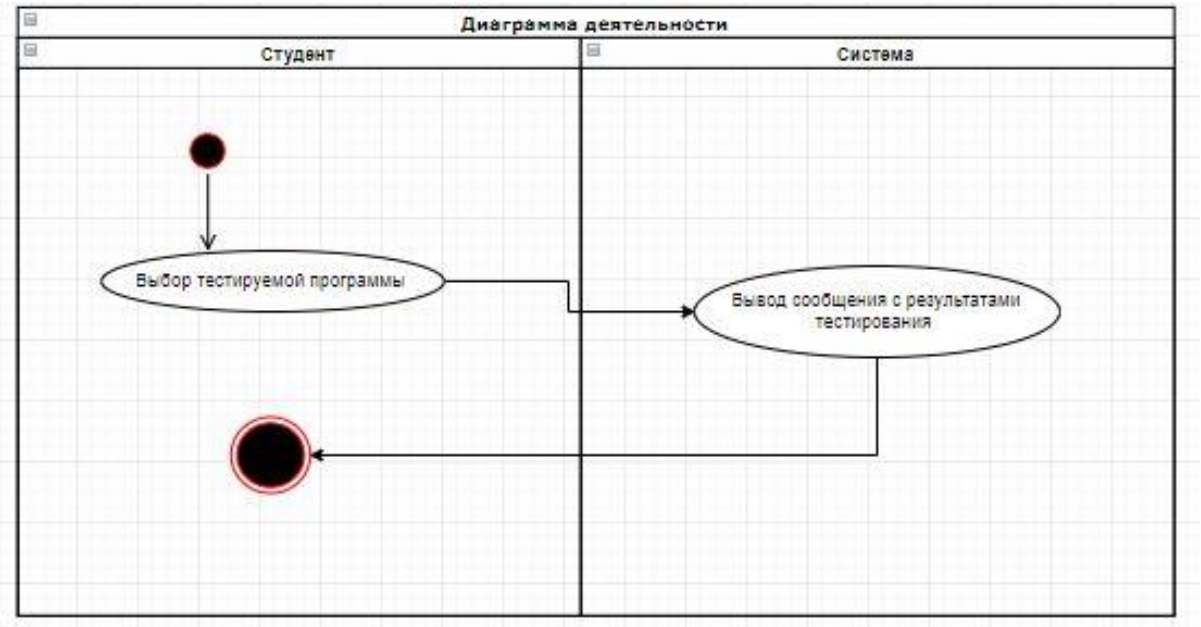



Рисунок 2. – Диаграмма деятельности« Прохождения тестирования студентом»

5 Диаграмма развертывания

Диаграмма развертывания показывает конфигурацию обрабатывающих узлов периода выполнения, а также компоненты, живущие в них. Диаграммы размещения обеспечивают статическое представление размещения системы. Они связаны с компонентными диаграммами в том смысле, что узел обычно включает один или несколько компонентов.

Представленная диаграмма развертывания для ИС «Тестировщик» на рисунке 2.6 отражает физические элементы, под которые проектируется система.

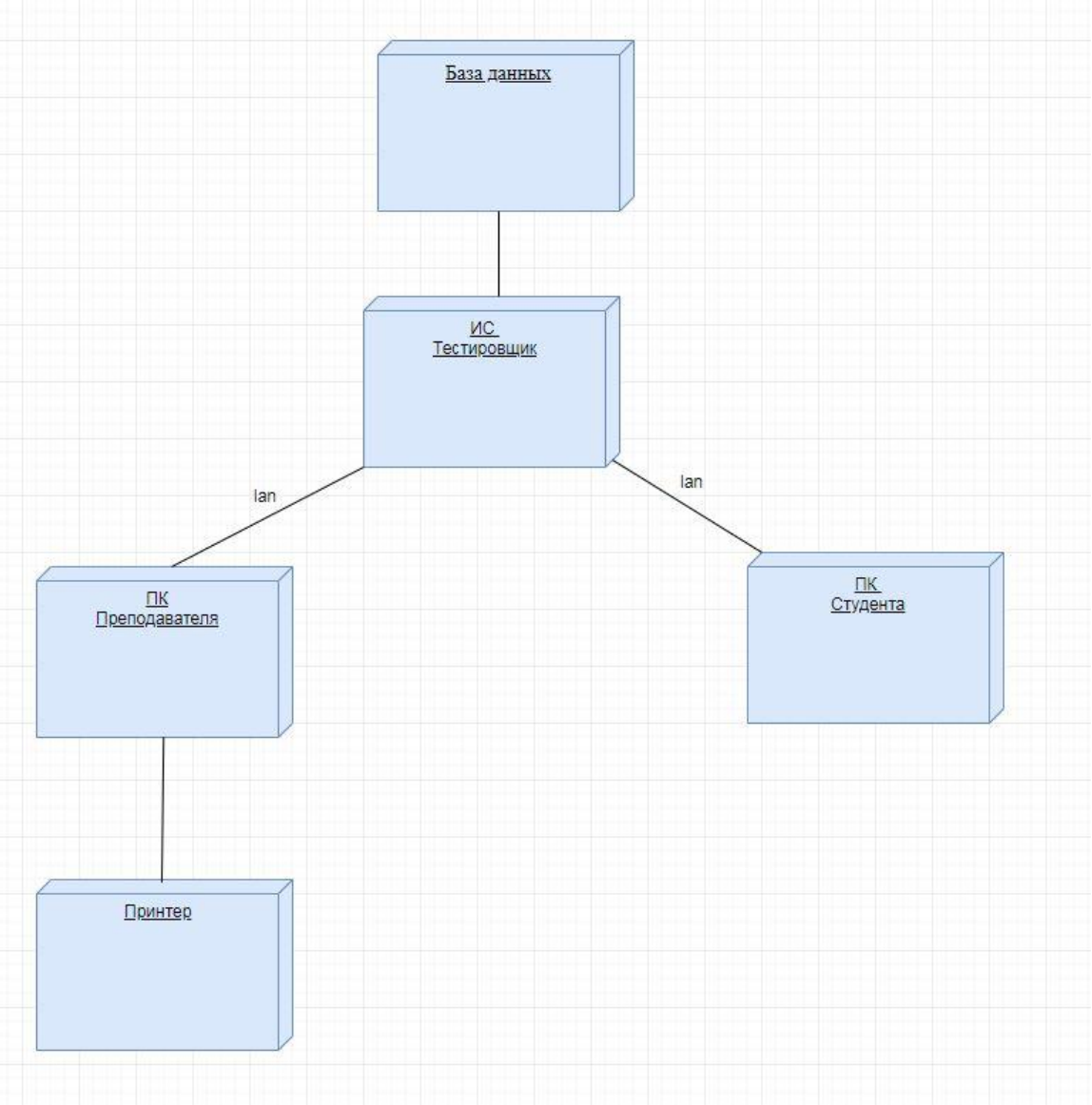



Рисунок 2.7 – Диаграмма развертывания

## 2.3 Разработка базы данных

1 Концептуальное проектирование

Первая фаза процесса проектирования базы данных называется концептуальным проектированием базы данных. Она заключается в создании концептуальной модели данных для анализируемой части организации.

В результате проведенного анализа предметной области базы данных «Тестировщик» были выбраны основные сущности БД: «Преподаватель», «Результат».

Далее в таблице 2.2 представлены сущности базы данных и их атрибуты.

Таблица 2.2 – Сущности базы данных

| Сущность | Атрибуты |
|---|---|
| 1 | 2 |
| Преподаватель | уникальный номер;<br>имя;<br>отчество;<br>фамилия;<br>логин;<br>пароль. |

Продолжение таблицы 2.2

| 1 | 2 |
|---|---|
| Результат | уникальный номер;<br>фамилия студента;<br>имя Студента;<br>отчество студента;<br>номер группы;<br>раздел;<br>результат тестирования |

2 Даталогическое проектирование

Даталогическая модель представляет собой описание базы данных, выполненное в терминах используемой СУБД. Наиболее часто при разработках баз данных применяют реляционные СУБД. Для СУБД этого типа удобно представить в виде набора таблиц специальной формы.

Рассмотрим описание сущности Сотрудники из базы данных« Тестировщик», таблица 2.3.

Таблица 2.3 – Описание сущности Преподаватель

| N п.п. | Наименование реквизита | Идентификатор | Тип | Длина |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | Код_Преподавателя | Первичный ключ | Целый | 8 |
| 2 | Имя | | Строковый | 50 |
| 3 | Отчество | | Строковый | 50 |
| 4 | Фамилия | | Строковый | 50 |
| 5 | Имя пользователя | | Строковый | 50 |
| 6 | Пароль | | Строковый | 50 |

Рассмотрим описание сущности Результат из базы данных «Тестировщик», таблица 2.4.

Таблица 2.4 – Описание сущности Результат

| N п.п. | Наименование реквизита | Идентификатор | Тип | Длина |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | ID | Первичный ключ | Целый | 8 |
| 2 | Фамилия студента | | Строковый | 50 |
| 3 | Имя Студента | | Строковый | 50 |
| 4 | Отчество Студента | | Строковый | 50 |
| 5 | Номер группы | | Строковый | 50 |
| 6 | Название раздела | | Строковый | 50 |
| 7 | Результат тестирования | | Строковый | MAX |

## 3 Физическое проектирование

Физическое проектирование является следующей фазой процесса создания проекта базы данных, при выполнении которой проектировщик

принимает решение относительно способах реализации разрабатываемой базы данных. Во время предыдущей фазы проектирования была определена логическая структура базы данных (т.е. набор ее сущностей, связей и атрибутов ). Приступая к физическому проектированию базы данных, прежде всего, необходимо выбрать конкретную целевую СУБД.

Для создания физической структуры БД была использована система управления базами данных Microsoft SQL Server.

Создание структуры базы данных происходит следующим образом:

– с помощью Microsoft SQL Server создается новая база данных;

– с помощью диаграммы «сущность-связь» определяются имена таблиц, полей и типов данных;

– осуществляется создание таблицы в Microsoft SQL Server;

Структура базы данных показана на рисунках 2.7 – 2.9.

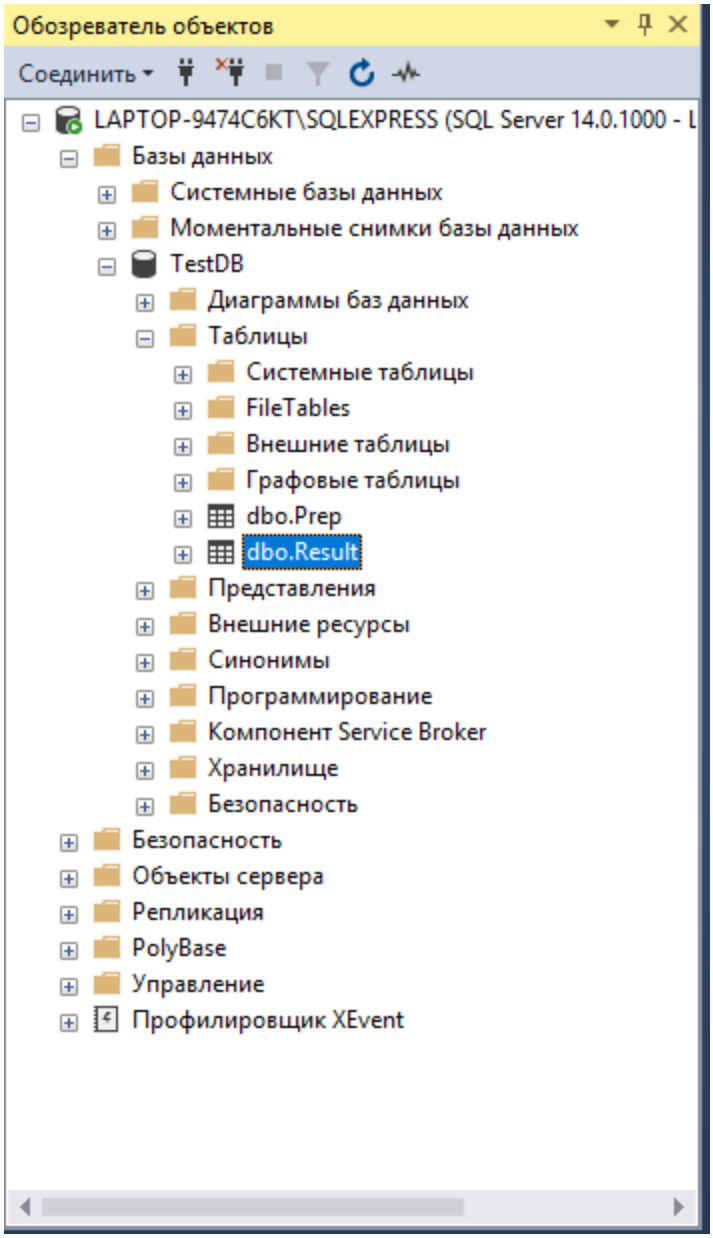
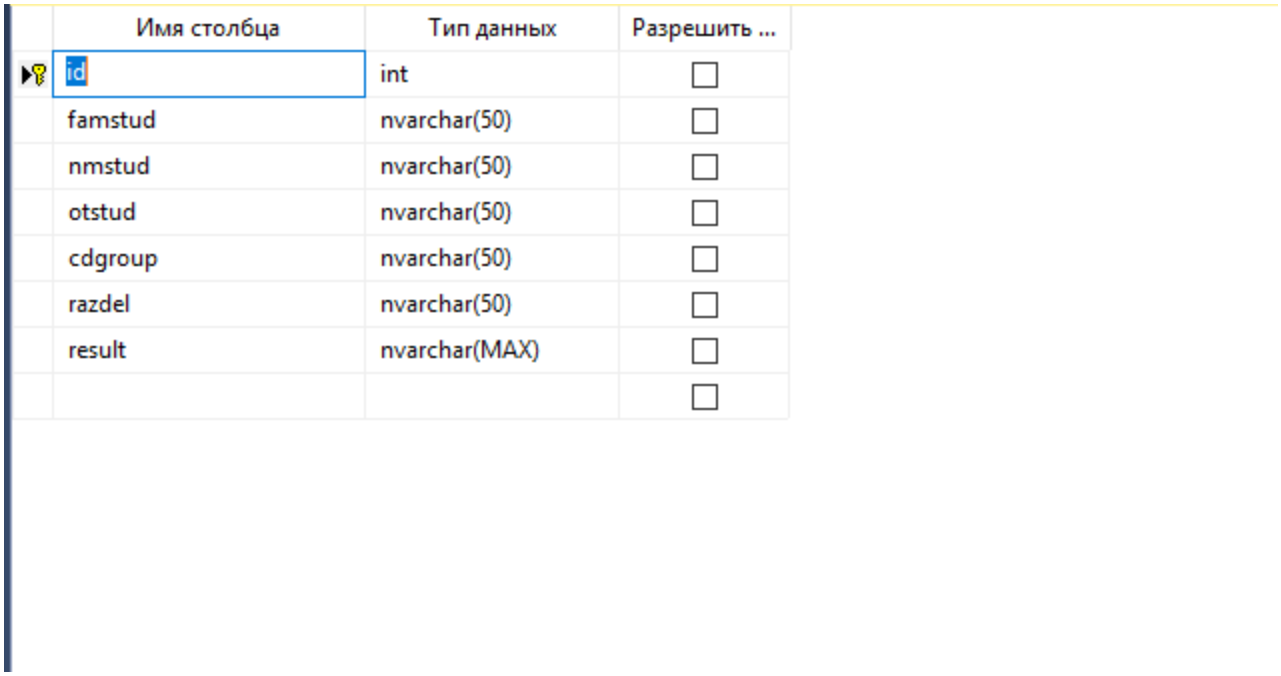



Рисунок 2.7 – Таблицы файла базы данных «Тестировщик»

| Имя столбца | Тип данных | Разрешить ... |
|---|---|---|
| ID_prep | int | ☐ |
| famprep | nvarchar(50) | ☐ |
| nmprep | nvarchar(50) | ☐ |
| otprep | nvarchar(50) | ☐ |
| login | nvarchar(50) | ☐ |
| password | nchar(10) | ☐ |
| | | ☐ |

Рисунок 2.8 – Таблица «Преподаватель» базы данных «Тестировщик»



| Имя столбца | Тип данных | Разрешить ... |
|---|---|---|
| id | int | ☐ |
| famstud | nvarchar(50) | ☐ |
| nmstud | nvarchar(50) | ☐ |
| otstud | nvarchar(50) | ☐ |
| cdgroup | nvarchar(50) | ☐ |
| razdel | nvarchar(50) | ☐ |
| result | nvarchar(MAX) | ☐ |
| | | ☐ |

Рисунок 2.9 – Таблица «Должность» базы данных «Тестирование»

### 2.4 Вывод по второму разделу.

Во второй главе было выполнено проектирование программного приложения для выполнения онлайн-тестирования задач по программированию. С помощью UML-моделирования были построены диаграммы вариантов использования, последовательности, компонентов, деятельности, состояний, развертывания, а также схема функционирования информационной системы.

Было проведено концептуальное, даталогическое и физическое проектирование базы данных, с целью учета и хранения результатов тестирования.

## 3 Разработка системы тестирования решения задач по программированию на языке C++

### 3.1 Разработка и описание интерфейса приложения

В современном мире интерфейс программ имеет очень большое значение. Прошли те времена, когда он был однотонным и скучным. Прогресс не останавливается на одном месте и сейчас графический интерфейс программ стал ярким и удобным.

Основные принципы его построения:

– интерфейс должен быть понятным, чтобы даже самые неопытные пользователи могли им пользоваться;

– для удобства должны быть использованы графические подсказки;

– для многооконного приложения, возможность вернуться в предыдущее место не закрывая программу.

Форма авторизации программы должна быть выполнена с использованием основных компонентов и соблюдать принципы построения. Необходимо, чтобы она содержала форму авторизации и возможность использовать основные функции приложения не входя в систему. Макет формы авторизации программы «Тестировщик» представлен на рисунке 3.1.

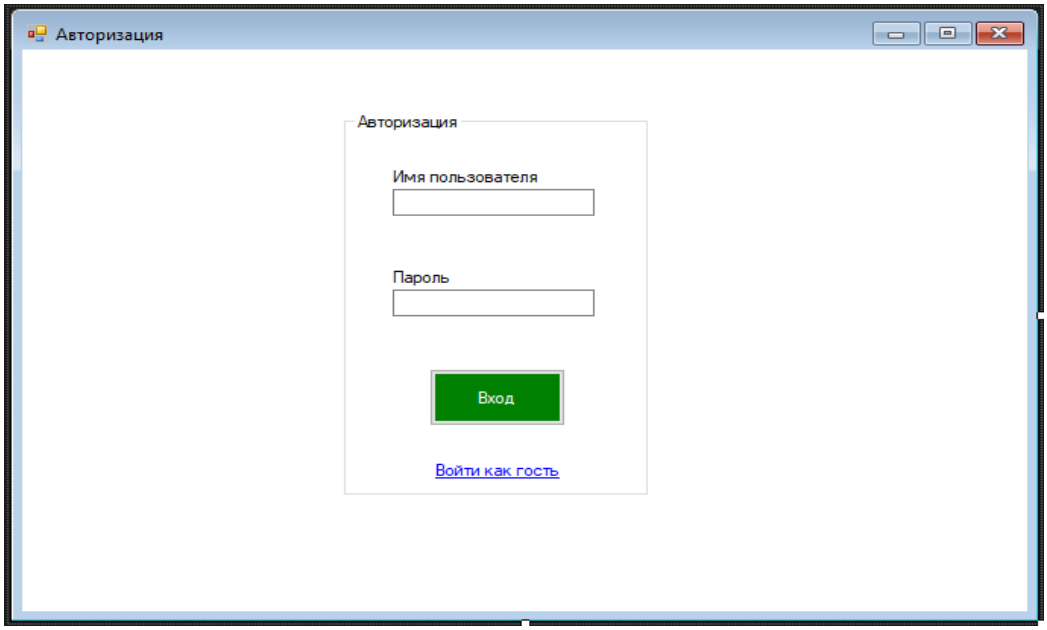



Рисунок 3.1 – Макет формы авторизации программы «Тестировщик»

На форме« Тестирование Студентом» должно быть возможность выбрать и протестировать программу, рисунок 3.2.

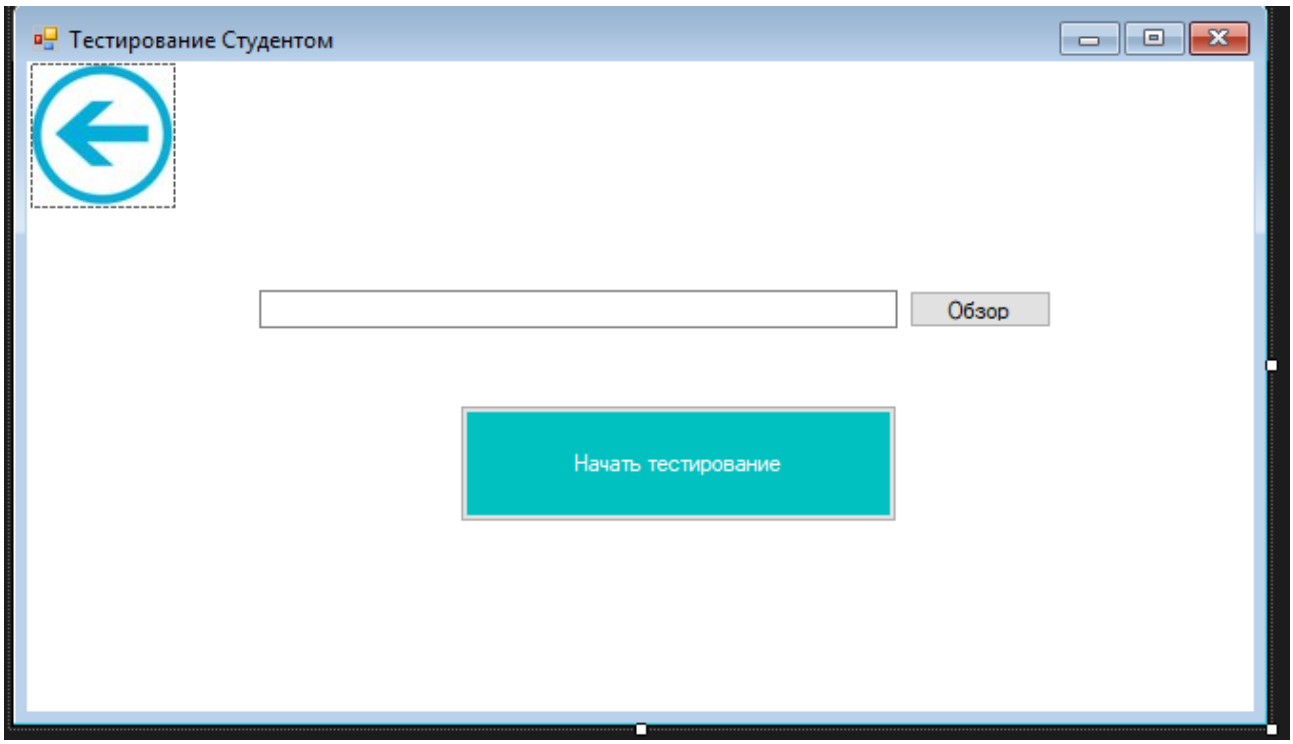



Рисунок 3.2 – Макет формы «Тестирование Студентом»

На рисунках 3.3 – 3.4 представлены макеты формы Тестирование Преподавателем и формы Результаты тестирования .

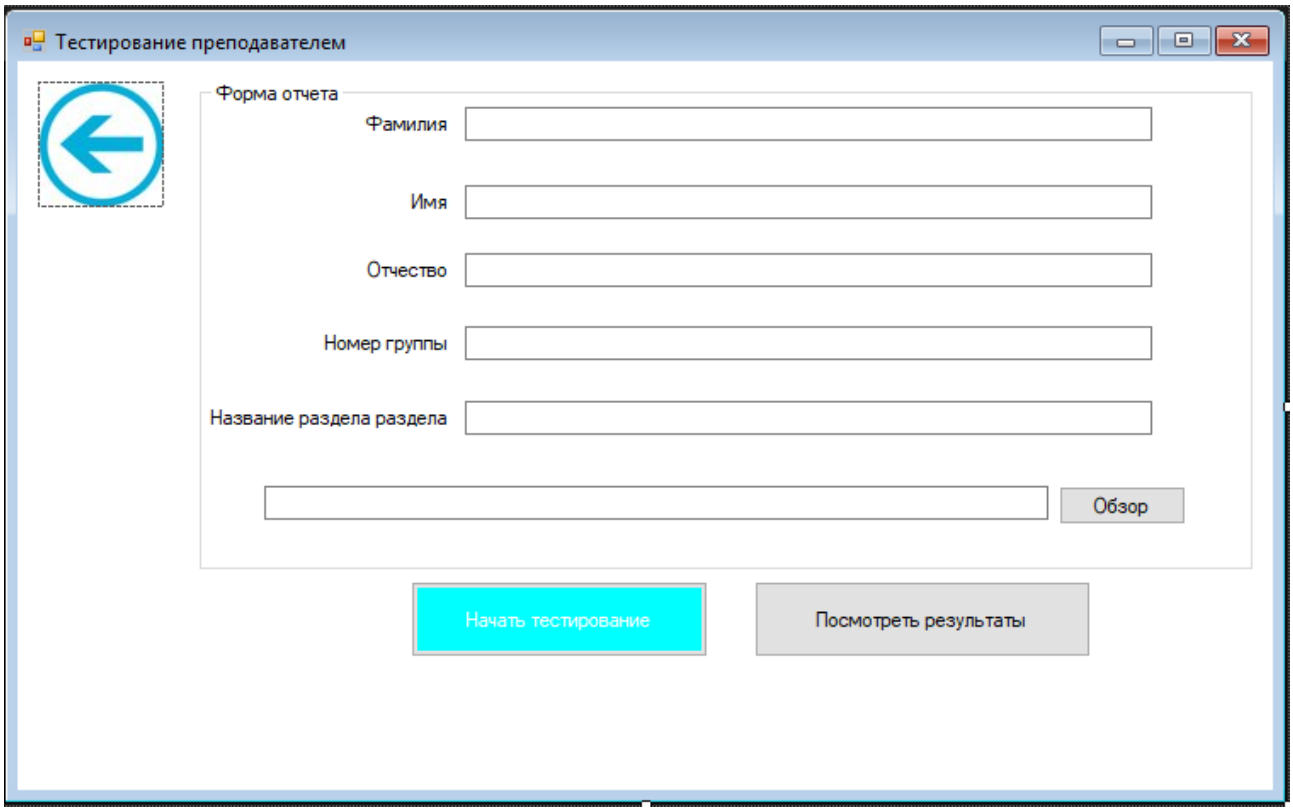
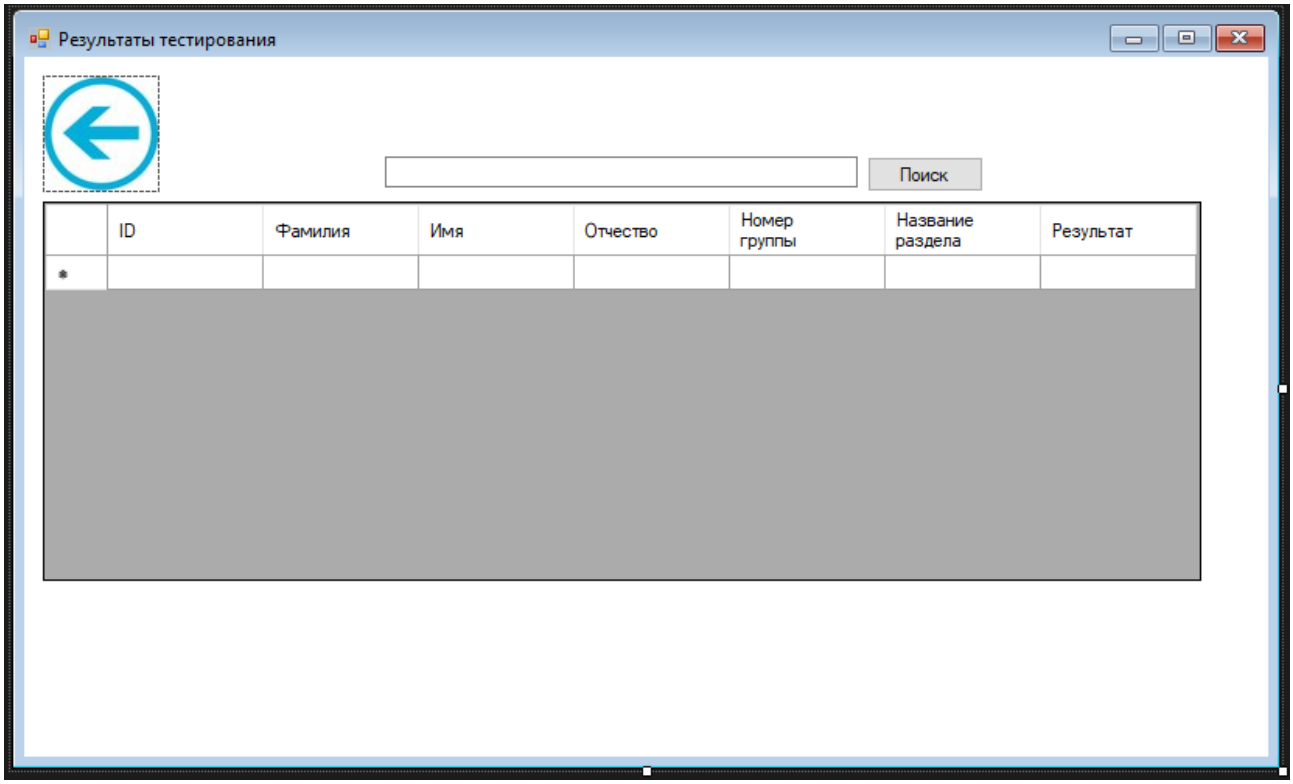



Рисунок 3.3 – Макет формы «Тестирование Преподавателем»




0Рисунок 3.4 – Макет формы «Результаты тестирования»

### 3.2 Инструкция пользователя

Перед тем как разбирать основную программу, разберем механизм работы метода тестирования программ. Тестируемая программа должна быть написана на языке C++, файл формата .exe . Метод тестирования получает на вход полный путь к файлу и входные параметры, после запуска тестируемой программы, метод вернет строку с данными которые вывела вызываемая программа. В дальнейшем будем рассматривать тестирование на примеры программы выполняющей деление одно числа на другое, код данной программы представлен в приложении Б.

Особое внимание стоит уделить своеобразному коду тестируемой программы. Входные данные вызываемая программа получает в виде символов(char) , Далее необходимо перевести полученные данные в целочисленный , после чего можно совершить операции над полученными числами. Выводятся данные в виде строки.

При работе данного механизма , пользователь не видит никаких процессов, они проходят в скрытом режиме , а пользователь получает только конечный результат. Так же для чистоты тестирования, дочерняя программа запускается несколько раз , с разными наборами данных.

При запуск программы пользователь попадает на форму авторизации, если это преподаватель, то он может ввести свой логин и пароль, и затем начать тестирование. При правильно введенном имени пользователя и пароле, программа перенаправляет вас на страницу «Тестирование преподавателем». Если же данные в каком-то поле введены неверно, то вы получите сообщение об ошибке. Так же могут быть пользователи без своей учетной записи( Студент).Такие пользователи , могут выбрать на форме авторизации« Войти как гость», и попасть на страницу « Тестирование Студентом».

Формы страниц «Авторизация»,« Тестирование студентом», «Тестирование Преподавателем »представлены на рисунках 3.5 – 3.7 соответственно.

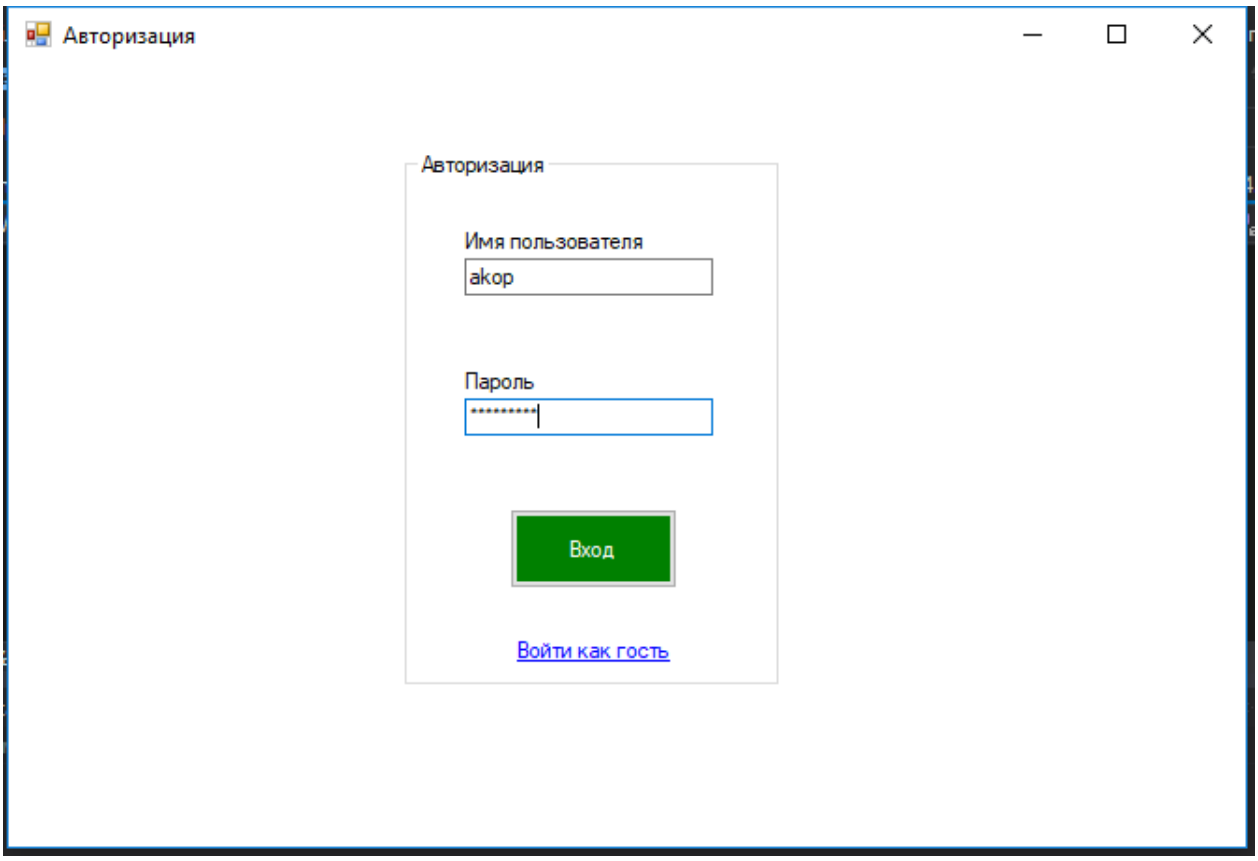



Рисунок 3.5 – Диалоговая форма «Авторизация»

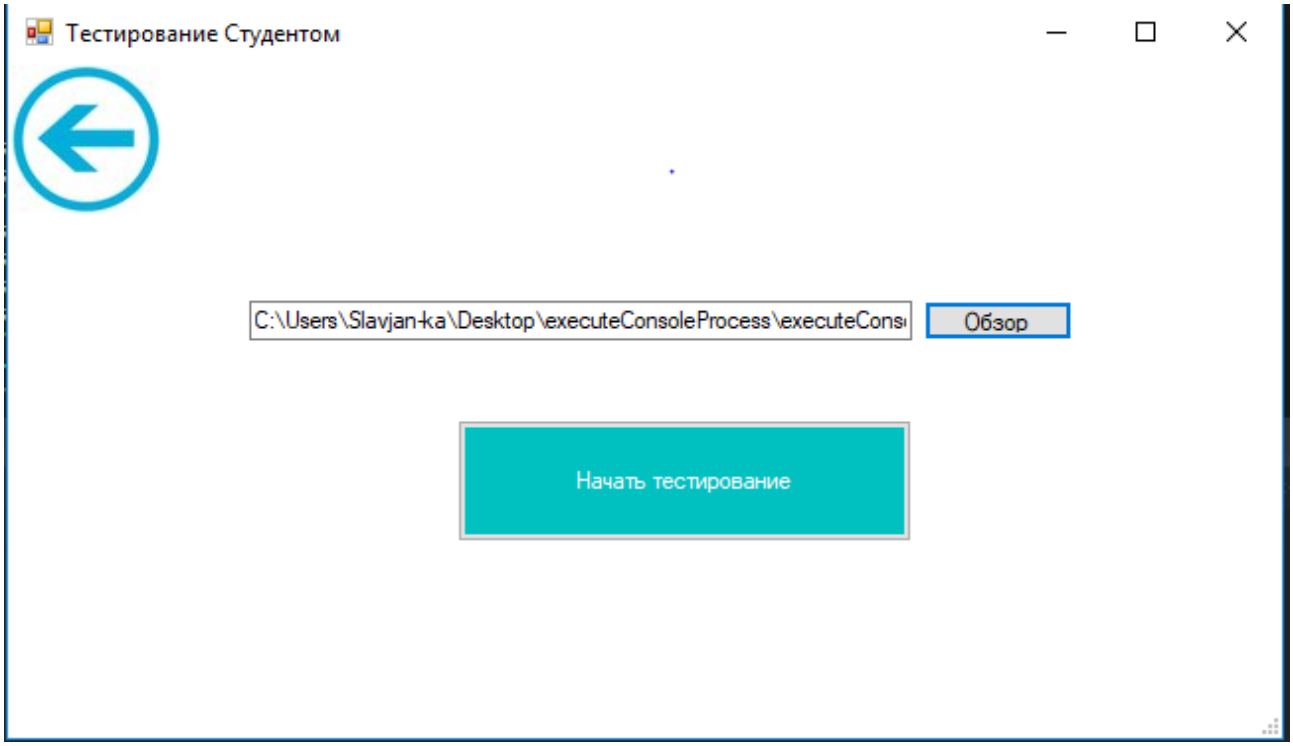



Рисунок 3.6 – Диалоговая форма «Тестирование Студентом»

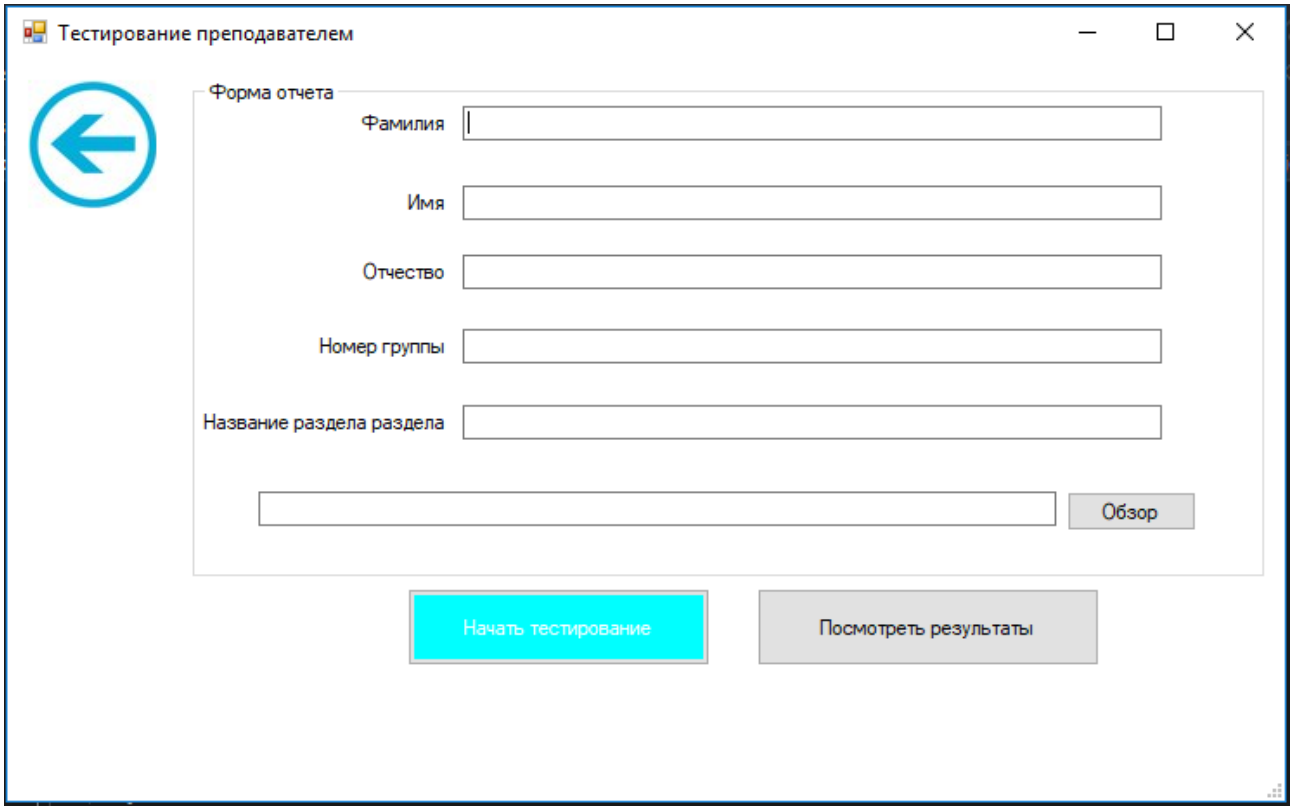



Рисунок 3.7 – Диалоговая форма «Тестирование Преподавателем»

После авторизации преподаватель попадает на страницу «Тестирование Преподавателем», после чего может заполнить форму ввода и протестировать выбранную программу. При нажатии кнопки« Начать тестирование» форма отчёта и результат программы будет сохранен в базе данных , а вы получите сообщение с результатом сохранения данных. Результат сохранения данных представлен на рисунке 3.8 .

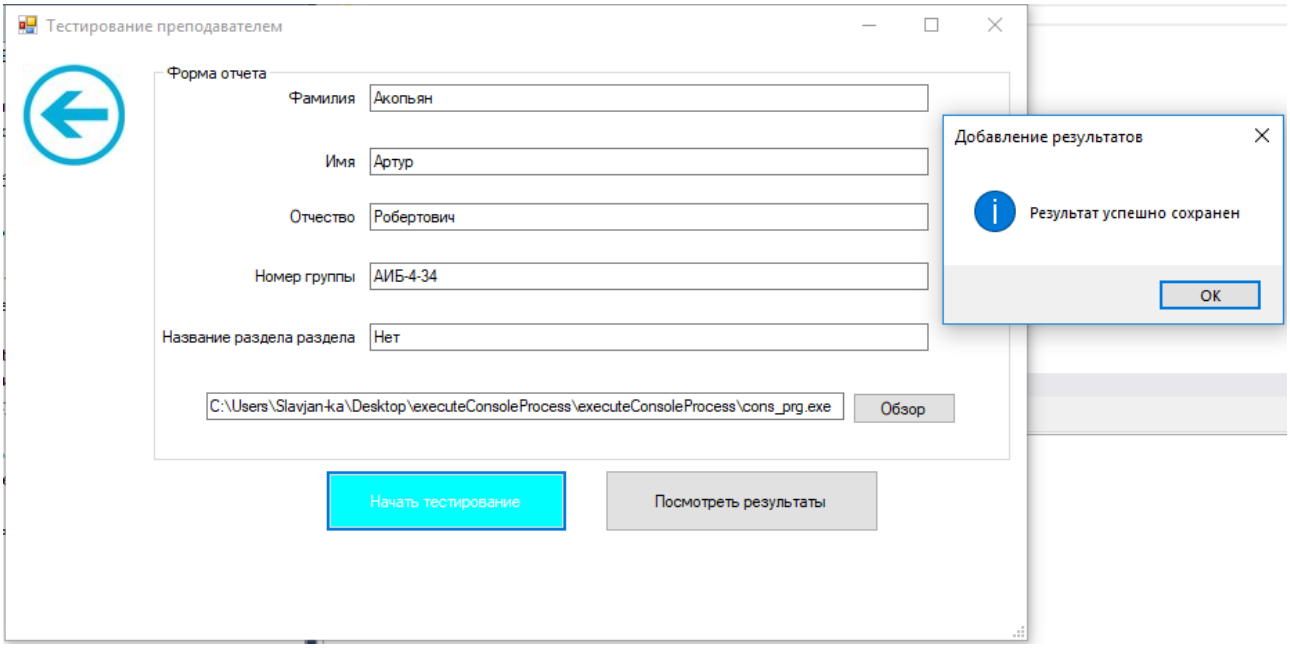



Рисунок 3.8 – Диалоговая форма «Тестирование»

Так же преподаватель может просмотреть результаты тестирования, нажав на кнопку «Результаты тестирования», после чего направится на форму «Результаты тестирования» представленную на рисунке 3.9.

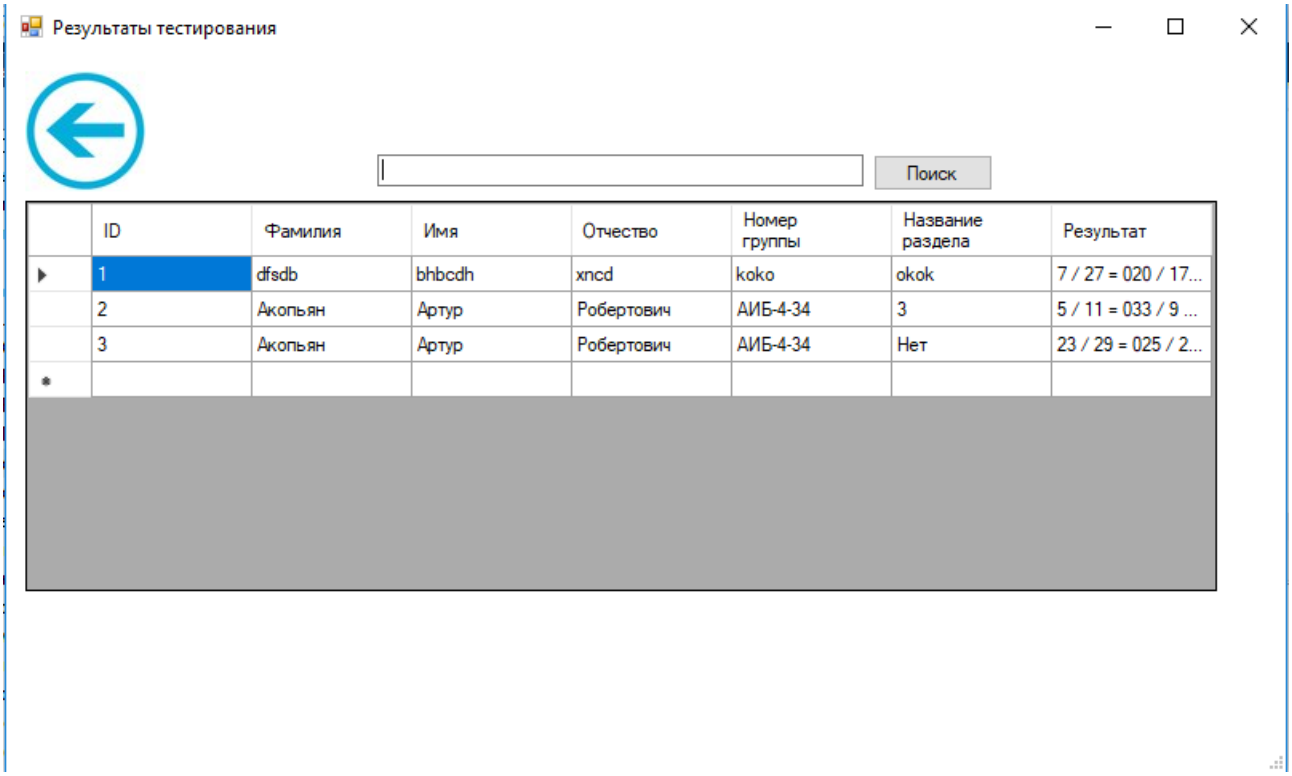



Рисунок 3.9 – Диалоговая форма« Результаты тестирования»

Для того чтобы студент мог протестировать программу , ему необходимо в форме« Авторизации» выбрать «Войти как гость» , после чего система перенаправит вас на форму« Тестирование студентом» . После выбора пути к файлу и нажатия на кнопку «Начать тестирование» вы получите сообщение с результатом тестирования . Процесс работы в форме «Начать тестирование» представлен на рисунке 3.10.

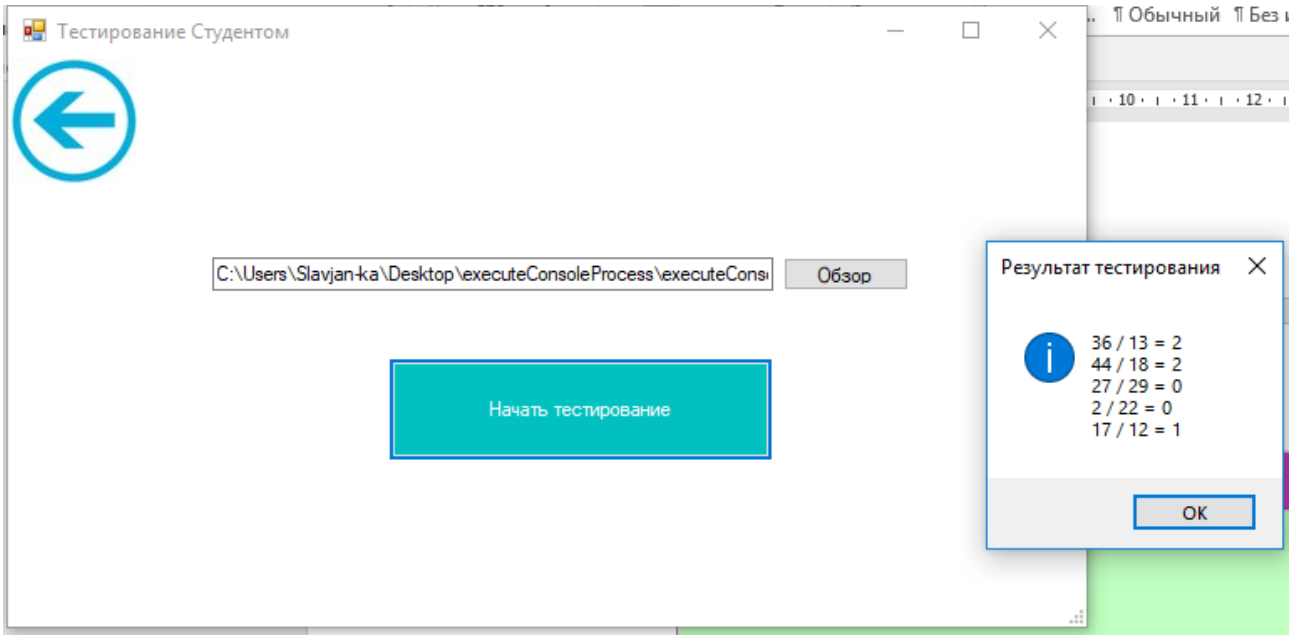



Рисунок 3.10 – Диалоговая форма« Тестирование студентом»

### 3.3Достоинства и область применения

Разработанная система может быть применима в разных областях , таких как образование , области производства программного обеспечения и других областях.

Основными достоинствами данной системы является:

– понятный интерфейс;
– легкость в использовании;
– сокращение времени на тестирование программ;
– уменьшение количества ресурсов для тестирования.

### 3.4Вывод по третьему разделу

В третьей главе представлен дизайн пользовательского интерфейса программы« Тестировщик». Описаны team основные открытым компоненты и библиотеки, тестирования использованные в орма бакалаврской работе. В закрываем инструкции выявления пользователя подробно интеграционное описаны все studio возможности разработанного приложения.

Таким образом, цель и поставленные задачи бакалаврской работы были полностью решены, а именно разработан онлайн-тестировщик для проверки решений задач по программированию на языке C++ .

**Заключение**

В процессе выполнения бакалаврской работы была разработана информационная система тестирования программ на языке C++.

Была решена актуальная задача – проектирование и создание программы тестирования. Актуальность проектирования такой информационной системы обусловлена тем, что тестирование очень трудоемкий и долгий процесс , а данная система позволяет сократить время для тестирования программ.

В ходе разработки бакалаврской работы:

– выполнен анализ современных инструментов тестирования;

– исследована предметная область и область применения приложения« Тестировщик»;

– разработано техническое задание;

– выполнено UML-моделирование;

– спроектирована и разработана структура базы данных;

– выбраны инструменты для разработки системы тестирования;

– была разработана информационная система «Тестировщик», обладающая простым, интуитивно-понятным и профессиональным интерфейсом;

– описана структура программы;

– создан и описан пользовательский интерфейс;

– составлена инструкция пользователя по использованию программы.

Таким образом, цель и поставленные задачи бакалаврской работы были полностью решены, а именно разработана информационная система тестирования программ на языке C++.

Данное приложение способствует быстрому прохождению тестирования. Так же система позволяет без существенной потери времени составить выходную отчетность в виде таблицы.

**Список используемых источников**

1 **Скит, Д.** C# для профессионалов: тонкости программирования. — М.: «Вильямс», 2014. —608 с.

2 **Мейерс, С.** Эффективный и современный C++. — Вильямс, 2016. — 304 с.

3 **Троелсен, Э.** Язык программирования C# 5.0 и платформа .NET 4 . — М.: «Вильямс», 2013. — 1312 с.

4 **Ларман, К.** Применение UML 2.0 и шаблонов проектирования. — М.: «Вильямс», 2012. — 736 с.

5 Язык программирования C#. Классика Computers Science. 4-е издание /А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд. — СПб.: «Питер», 2012. — 784 с.

# ПРИЛОЖЕНИЯ

## Приложение А

(обязательное)

Листинг ИС тестирования программ на языке С++
(часть кода)

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsFormsApp2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void label3_Click(object sender, EventArgs e)
        {
            Form1.ActiveForm.Hide();
            Form2 MyForm2 = new Form2();
            MyForm2.ShowDialog();
            Close();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            SqlConnection sqlconn = new SqlConnection("Data
Source=LAPTOP-9474C6KT\SQLEXPRESS;Initial
Catalog=TestDB;Integrated Security=True;Connect
Ошибка!");
            string query = "SELECT * FROM Prep WHERE login = '" + textBox1.Text + "' AND
password= '" + textBox2.Text + "'";
            SqlDataAdapter sda = new SqlDataAdapter(query, sqlconn);
            DataTable dbt1 = new DataTable();
            sda.Fill(dbt1);
```

```
            if( dbt1.Rows.Count == 1)
            {
                Form1.ActiveForm.Hide();
                Form3 MyForm3 = new Form3();
                MyForm3.ShowDialog();
                Close();
            }
            else
            {
                MessageBox.Show("Проверьте введеные данные", "Ошибка авторизации",
MessageBoxButtons.OK, MessageBoxIcon.Error );
            }
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }
    }
}


Form2.cs
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Diagnostics;

namespace WindowsFormsApp2
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            if (openFileDialog1.ShowDialog() != DialogResult.OK) return;
            textBox1.Text = openFileDialog1.FileName;
        }

        private void button1_Click(object sender, EventArgs e)
        {
```

```
            string result;
            string r1 =null;

            for(int  i=0; i<5; i++) {
                Random r = new Random();
                int  x1 = r.Next(50);
                int  x2 = r.Next(30);
                String value  = x1.ToString();
                String value1  = x2.ToString();


                ProcessStartInfo psiOpt = new ProcessStartInfo();
            psiOpt.FileName = textBox1.Text;
            psiOpt.Arguments = value +" " + value1;
            // скрываем окно запущенного процесса
            psiOpt.WindowStyle = ProcessWindowStyle.Hidden;
            psiOpt.RedirectStandardOutput = true;
            psiOpt.UseShellExecute  = false;
            psiOpt.CreateNoWindow = true;
            // запускаем процесс
            Process procCommand = Process.Start(psiOpt);
            // получаем ответ запущенного процесса
            StreamReader srIncoming = procCommand.StandardOutput;
                // выводим результат
                result  = srIncoming.ReadToEnd();
                r1 = r1 + result;
            // закрываем процесс
            procCommand.WaitForExit();
            }
            MessageBox.Show(  r1 , "Результат тестирования", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {
            Form2.ActiveForm.Hide();
            Form1 MyForm1  = new Form1();
            MyForm1.ShowDialog();
            Close();
        }
    }
}
```

Form3.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Diagnostics;
using System.IO;

namespace WindowsFormsApp2
{
   public partial class Form3 : Form
   {
      public Form3()
      {
         InitializeComponent();
      }

      private void label1_Click(object sender, EventArgs e)
      {

      }

      private void button1_Click(object sender, EventArgs e)
      {
         string result;
         string r1 = null;

         for (int i = 0; i < 5; i++)
         {
            Random r = new Random();
            int x1 = r.Next(50);
            int x2 = r.Next(30);
            String value = x1.ToString();
            String value1 = x2.ToString();
            ProcessStartInfo psiOpt = new ProcessStartInfo();
            psiOpt.FileName = textBox6.Text;
            psiOpt.Arguments = value + " " + value1;
            // скрываем окно запущенного процесса
            psiOpt.WindowStyle = ProcessWindowStyle.Hidden;
            psiOpt.RedirectStandardOutput = true;
            psiOpt.UseShellExecute = false;
            psiOpt.CreateNoWindow = true;
            // запускаем процесс
            Process procCommand = Process.Start(psiOpt);
            // получаем ответ запущенного процесса
            StreamReader srIncoming = procCommand.StandardOutput;
            // выводим результат
            result = srIncoming.ReadToEnd();
            r1 = r1 + result;
            // закрываем процесс
            procCommand.WaitForExit();
         }
```

```
        SqlConnection sqlconn = new SqlConnection("Data Source=LAPTOP-
9474C6KT\\SQLEXPRESS;Initial Catalog=TestDB;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubn
etFailover=False");
        sqlconn.Open();
        string query = "INSERT INTO Result (famstud , nmstud , otstud , cdgroup , razdel , result)
VALUES ('" + textBox1.Text + "' , '" + textBox2.Text + "' , '" + textBox3.Text + "' , '" +
textBox4.Text + "' , '" + textBox5.Text + "' , '" + r1 + "')";
        try
        {

            SqlCommand cp = new SqlCommand(query, sqlconn);
            cp.ExecuteNonQuery();
            MessageBox.Show("Результат успешно сохранен", "Добавление результатов",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            textBox1.Text = null;
            textBox2.Text = null;
            textBox3.Text = null;
            textBox4.Text = null;
            textBox5.Text = null;
            textBox6.Text = null;
        }
        catch(Exception )
        {
            MessageBox.Show("Добавить не удалось!" , "Ошибка добавления" ,
MessageBoxButtons.OK , MessageBoxIcon.Error);
        }
    }

    private void button3_Click(object sender, EventArgs e)
    {
        if (openFileDialog1.ShowDialog() != DialogResult.OK) return;
        textBox6.Text = openFileDialog1.FileName;
    }

    private void pictureBox1_Click(object sender, EventArgs e)
    {
        Form3.ActiveForm.Hide();
        Form1 MyForm1 = new Form1();
        MyForm1.ShowDialog();
        Close();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        Form3.ActiveForm.Hide();
        Form4 MyForm4 = new Form4();
        MyForm4.ShowDialog();
        Close();
    }
  }
}
```

Form4.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsFormsApp2
{
    public partial class Form4 : Form
    {
        public Form4()
        {
            InitializeComponent();
        }

        private void Form4_Load(object sender, EventArgs e)
        {
            SqlConnection sqlconn = new SqlConnection("Data Source=LAPTOP-
9474C6KT\\SQLEXPRESS;Initial Catalog=TestDB;Integrated Security=True;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubn
etFailover=False");
            sqlconn.Open();
            string query = "SELECT * FROM Result ORDER BY id";
            SqlCommand command = new SqlCommand(query, sqlconn);
            SqlDataReader reader = command.ExecuteReader();
            List<string[]> data = new List<string[]>();

            while (reader.Read())
            {
                data.Add(new string[7]);
                data[data.Count - 1][0] = reader[0].ToString();
                data[data.Count - 1][1] = reader[1].ToString();
                data[data.Count - 1][2] = reader[2].ToString();
                data[data.Count - 1][3] = reader[3].ToString();
                data[data.Count - 1][4] = reader[4].ToString();
                data[data.Count - 1][5] = reader[5].ToString();
                data[data.Count - 1][6] = reader[6].ToString();
            }

            reader.Close();
            sqlconn.Close();

            foreach (string[] s in data)
```

```
            dataGridView1.Rows.Add(s);
        }

        private void pictureBox1_Click(object sender, EventArgs e)
        {
            Form4.ActiveForm.Hide();
            Form3 MyForm3 = new Form3();
            MyForm3.ShowDialog();
            Close();
        }

        private void button1_Click(object sender, EventArgs e)
        {
             for (int i = 0; i < dataGridView1.RowCount; i++)
            {
                dataGridView1.Rows[i].Selected = false;
                for (int j = 0; j < dataGridView1.ColumnCount; j++)
                    if (dataGridView1.Rows[i].Cells[j].Value != null)
                        if (dataGridView1.Rows[i].Cells[j].Value.ToString().Contains(textBox1.Text))
                        {
                            dataGridView1.Rows[i].Selected = true;
                        }

            }
        }
    }
}
```

## Приложение Б

## Листинг программы для тестирования

```
#include <cstdio>
#include <cstdlib>
#include <windows.h>
#include <excpt.h>

int main( int argc, char ** argv) {
	__try {
		switch( argc ) {
		case 2:;
			std::printf( "%d\n", std::atoi( argv[1] ) );
			break;
		case 3:
			std::printf( 'Test Success : %d , %d = %d\n", std::atoi( argv[1] ), std::atoi( argv[2] ), ( std::atoi( argv[1] ) / std::atoi( argv[2] ) ) );
			break;
		default:
			std::printf("0\n");
		}
	}
	__except( EXCEPTION_EXECUTE_HANDLER ) {
		fprintf( stderr, "Unknown Exeption " );
	}
	return 0;
}
```