**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc185494475)

[1 Постановка задачи 5](#_Toc185494476)

[1.1 Анализ предметной области 5](#_Toc185494477)

[2 Техническое задание на разработку ГОСТ 19.201-78 6](#_Toc185494478)

[2.1 Основания для разработки 6](#_Toc185494479)

[2.2 Назначение для разработки 6](#_Toc185494480)

[2.3 Требования к программе или программному изделию 6](#_Toc185494481)

[2.4 Паспорта модулей и метрики 8](#_Toc185494482)

[3 Проектирование диаграмм 13](#_Toc185494483)

[3.1 Поведенческие диаграммы 13](#_Toc185494484)

[3.2 Структурные диаграммы 16](#_Toc185494485)

[3.3 Модель С4 18](#_Toc185494486)

[4 Тестирование ПО 20](#_Toc185494487)

[4.1 Тестирование программы 20](#_Toc185494488)

[ЗАКЛЮЧЕНИЕ 21](#_Toc185494489)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 22](#_Toc185494490)

[ПРИЛОЖЕНИЕ 23](#_Toc185494491)

# **ВВЕДЕНИЕ**

**Учебная практика ориентирована на освоение процесса создания программного обеспечения, который включает в себя разработку документации, проектирование, реализацию, тестирование и анализ метрик ПО. Основной задачей является формулирование требований к программному обеспечению, разработка алгоритма и модели программы, а также создание функциональных модулей на основе спецификаций. В процессе работы необходимо обеспечить высокое качество проектирования диаграмм в соответствии с функциональными требованиями.**

**Документация должна содержать описание алгоритма работы программы, модель программы, исходный код с пользовательскими интерфейсами, протоколы тестирования и скриншоты результатов. Ключевыми этапами работы будут анализ проектной и технической документации, выявление ошибок в системных компонентах и использование встроенных инструментов для тестирования. Также потребуется провести анализ кода с целью его оптимизации и снятия метрик ПО для оценки качества программы.**

**Программа будет подвергнута тестированию для выявления ошибок и проверки её соответствия спецификациям. Документация, созданная в ходе практики, должна отражать весь процесс разработки — от анализа требований до получения окончательных результатов тестирования с метриками.**

# **1 Постановка задачи**

## 1.1 Анализ предметной области

Программа предназначена для работы с графиком акций, автоматически заполнять его значениями с использованием генератора случайных чисел (ГСЧ). Программа также должна выводить графическое представление данных.

Программа будет работать со случайных чисел и обеспечивать следующие основные функции.

Графическое отображение: Программа должна выводить графики:

* отображение графика;
* автоматически отрисосывывает график.

Интерфейс пользователя:

* программа автоматически генерирует случайные числа;
* отображаются графики для каждого из этапов.

Таким образом, программа предоставляет пользователю простой и интуитивно понятный интерфейс для работы с графиком, визуализации результатов в виде графиков, что способствует лучшему пониманию распределения чисел на графике.

# **2 Техническое задание на разработку ГОСТ 19.201-78**

## 2.1 Основания для разработки

Основанием для разработки является задание по учебной практике по Модулю ПМ.02 Осуществление интеграции программных модулей подписанное председателем цикловой комиссии А.Н. Юдаевыем 10.12.2024 г. и руководителем учебно-производственной практики и службы трудоустройства А.В. Суворовым 12.12.2024 г. Руководителем практики является Е.А. Мачнева.

## 2.2 Назначение для разработки

Назначение для разработки программы заключается в создании инструмента для работы с акциями, который будет автоматически генерировать элементы графика на основе случайного числа. Визуализация результатов будет выполнена в виде графиков, которые поочередно отображают на графике.

Программа предназначена для удобного анализа и обработки данных, улучшая понимание распределения чисел на графике и обеспечивая наглядное представление результатов через графическое отображение. Она может быть полезна для пользователей, которые занимаются анализом числовых данных и визуализацией, а также для изучающих работу с акциями и их графиками.

## 2.3 Требования к программе или программному изделию

Требования к обеспечению надежного функционирования программы не предъявляются.

**2.3.1 Надежность и восстановление после сбоев**

Требования к обеспечению надежного функционирования программы не предъявляются. Время восстановления после нефатального сбоя операционной системы не должно превышать 10 минут при соблюдении условий эксплуатации технических и программных средств. Время восстановления после фатального сбоя зависит от времени, необходимого для устранения неисправностей технических средств и переустановки программного обеспечения.

**2.3.2 Отказы из-за некорректных действий оператора**

Отказы программы могут возникнуть из-за некорректных действий оператора при взаимодействии с операционной системой. Для минимизации таких рисков рекомендуется ограничить пользователю доступ к административным привилегиям.

**2.3.3 Условия эксплуатации**

Программа должна работать в климатических условиях, удовлетворяющих требованиям, предъявляемым к техническим средствам (ПК) в части эксплуатации.

**2.3.4 Требования к обслуживанию**

Программа не требует проведения каких-либо видов обслуживания.

**2.3.5 Требования к численности и квалификации персонала**

Минимальное количество персонала для эксплуатации программы должно составлять 2 человека: системный администратор и пользователь ПК – оператор. Системный администратор должен иметь высшее профильное образование. Его задачи включают поддержание работоспособности технических средств и системных программных средств, включая операционную систему.

**2.3.6 Требования к техническим средствам**

Программа должна работать на IBM-совместимом персональном компьютере с следующими характеристиками:

* процессор: Pentium 4 2.5 GHz (одноядерный) или выше;
* свободное место на жестком диске: от 1 Гб;
* оперативная память (ОЗУ): 4 Гб и выше;
* операционная система: Windows 10.

**2.3.7 Требования к информационной и программной совместимости**

Информационные структуры и методы решения: Требования к информационным структурам (записям) на входе и методам решения не предъявляются. Информационная структура должна включать отображение входных данных и результата на экране.

Исходные коды и языки программирования: Исходные коды программы должны быть реализованы на языке C#. В качестве интеграционной среды разработки программы должна быть использована среда Microsoft Visual Studio 2022 и приложения для построения диаграмм Draw.Io.

Защита информации и программ: Требования к защите информации и программ не предъявляются.

**2.3.8 Специальные требования**

Программа должна обеспечивать взаимодействие с пользователем через графический интерфейс.

**2.3.9 Требования к программной докуменатции**

В состав сопровождающей документации должны входить:

* техническое задание на разработку (ГОСТ 19.201-78);
* руководство оператора (ГОСТ 19.505-79);
* программа, методика и результаты испытаний (ГОСТ 19.301-79).

## 2.4 Паспорта модулей и метрики

Разработка паспорта модуля и снятие метрик по заданию: Построить график динамики потребления электроэнергии (кв/ч) в колледже. График должен строиться постоянно, в качестве источника данных по потреблению электроэнергии использовать ГСЧ2 (должны учитываться рабочие часы).

**2.4.1 Спецификации для паспорта модуля**

Метод Timer\_Tick генерирует случайные числа со случайной размерностью.

Таблица 1 – Спецификация для функции Timer\_Tick

|  |  |  |
| --- | --- | --- |
| **Имя** | **Функция** | **Параметры** |
| Timer\_Tick | Генерация массива случайных чисел заданной размерности | Входные данные:  double. Выходные данные: Массив чисел, сгенерированный с помощью метода Random. |

При нажатии на кнопку, метод StartButton\_Click вызывает метод DrawGraph, передавая в него исходный случайные числа, который был сгенерирован методом Timer\_Tick.

Таблица 2 – Спецификация для функции StartButton\_Click

|  |  |  |
| --- | --- | --- |
| **Имя** | **Функция** | **Параметры** |
| ShowOriginalGraph | Отображение графика | Входные данные:  Нет параметров (график строится на основе сгенерированных чисел). Выходные данные: График. |

**2.4.2 Блок-схемы для паспорта модуля**

В данной блок-схеме (рисунок 1) мы видим работу кнопки «Генерировать числа», при нажатие на «Запустить» мы попадаем в функцию Timer\_Tick в которой происходит генерация чисел, по итогу выводится числа на графике.

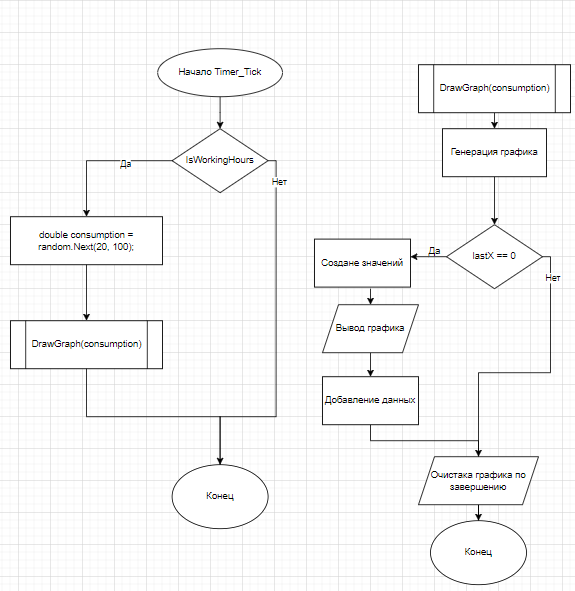


Рисунок 1 – Блок-схема для «Генерирования случайных чисел»

В блок-схеме (рисунок 1) мы видим работу кнопки «Запустить», при нажатии на кнопку пользователю выводится исходный график. Кнопка работает при помощи классов Timer\_Tick () и IsWorkingHours, функция Timer\_Tick () используется для генерации чисел, DrawGraph нужна для того чтобы отрисовывать графики.

**2.4.3 Метрики**

1. Индекс удобства поддержки: 82

* этот индекс показывает, насколько легко поддерживать и сопровождать код:
* значение 76 находится в хорошем диапазоне. Значения выше 20 считаются приемлемыми;
* 76–100: отлично, код легко поддерживать;
* 50–75: хорошо, но есть место для улучшений;
* <50: код сложно поддерживать; рекомендуется рефакторинг.

Заключение: Мой код имеет хороший индекс поддерживаемости.

1. Сложность организации циклов: 25

* измеряет количество возможных путей выполнения программы (ветвлений, условий if, else, циклов и т.д.);

Значение 28 указывает на умеренную сложность.

* 1–10: простая логика;
* 11–20: умеренно сложная логика;
* >20: сложный код; рекомендуется декомпозировать большие методы.

Заключение: Сложность немного выше нормы. Возможно, стоит разделить код на более мелкие методы для упрощения.

1. Глубина наследования: 9

* показывает глубину иерархии наследования. Чем глубже иерархия, тем сложнее понять и поддерживать код.

Глубина 9 — достаточно высокая.

* 0–3: нормально, низкая сложность;
* 4–6: средняя сложность;
* >6: высокая сложность; код становится менее гибким.

Заключение: Высокая глубина наследования может привести к проблемам с поддержкой и расширяемостью кода.

1. Взаимозависимость классов: 36

* измеряет, насколько класс зависит от других классов и библиотек;
* 0–10: низкая связность (хорошо);
* 11–20: умеренная связность;
* >20: высокая связность; код сильно зависит от внешних классов.

Заключение: Значение 9 указывает на низкую связность, что является хорошим показателем.

1. Строки исходного кода: 309

* общее количество строк в исходном коде (без учета пустых строк и комментариев).

Заключение: Код небольшой по объему, что упрощает его чтение и поддержку.

1. Строки исполняемого кода: 49

* количество строк, которые действительно выполняются при запуске программы.

Заключение: Код содержит 49 исполняемых строк, что соответствует небольшому объему и линейности программы.

1. Общее заключение:

* индекс удобства поддержки: хороший (82);
* сложность кода: умеренная (25) — рекомендуется декомпозировать сложные методы;
* глубина наследования: высокая (9) — стоит упростить иерархию классов;
* связность классов: низкая (36) — хороший показатель;
* размер кода: компактный и небольшой.

# **3 Проектирование диаграмм**

## 3.1 Поведенческие диаграммы

Основное внимание в данном контексте уделяется динамическим аспектам системы программного обеспечения или процесса. Эти диаграммы иллюстрируют функциональные возможности системы и демонстрируют, что должно происходить в моделируемой системе. Проще говоря, поведенческие диаграммы описывают действия объектов или субъектов. В этом типе нет строгого акцента на формальную разработку ПО. При создании поведенческой диаграммы всегда можно предоставить описание бизнес-процесса.

На рисунке 5 представлена диаграмма прецедентов, в которой описывается, что делает система, но не то, каким образом это осуществляется. Вариант использования представляет собой набор событий, которые происходят, когда «оператор» взаимодействует с системой для завершения процесса. Оператором может быть любой элемент, взаимодействующий с системой (человек, организация или приложение) извне системы. Таким образом, диаграмма вариантов использования визуально демонстрирует этот набор последовательностей и представляет функциональные требования системы.

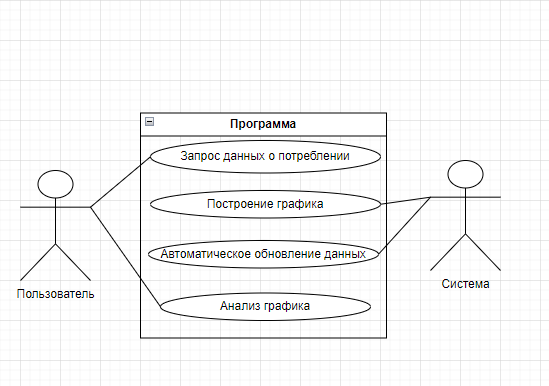


Рисунок 5 – Диаграмма прецедентов

На рисунке 6 мы видим диаграмму последовательности, диаграмма последовательности – это UML-диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актеров (действующих лиц) информационной системы в рамках прецедента.

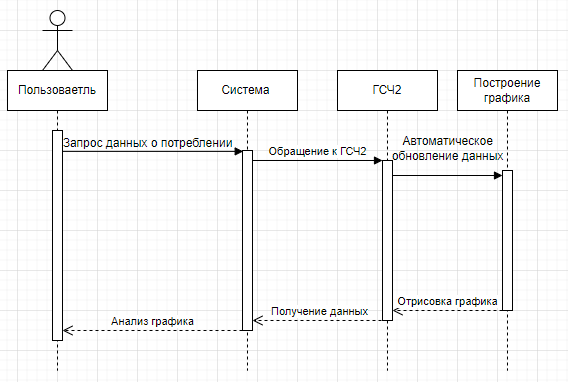


Рисунок 6 – Диаграмма последовательности

На рисунке 7 мы видим диаграмму активности (деятельности), это поведенческая диаграмма, позволяющая разделить действия и их последовательность по ролям. Данная графическая структура используется в анализе и оптимизации различных бизнес-процессов. Суть диаграммы активности в том, чтобы отобразить все необходимые ресурсы для решения поставленных задач. Следовательно, диаграмма деятельности необходима так же для оценки затрат разных показателей: временной (мин./сек.), финансовый (руб./тыс.руб.), единичный (шт.). Эта оценка поможет при поиске различных решений – взвесив при этом все за и против.

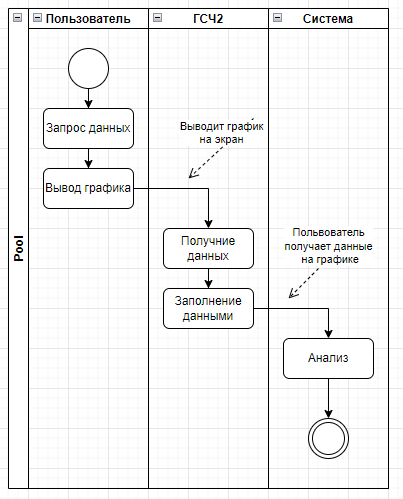


Рисунок 7 – Диаграмма активности

На рисунке 8 мы видим диаграмму состояний, это еще одна поведенческая диаграмма, которая описывает все возможные состояния системы, объекта или объекта внутри системы. Ее главная цель – графически отобразить все этапы различных систем и ее участников. Данная диаграмма широко используется при анализе и проектировании систем, где важно моделировать поведение объектов (или систем) в различных сценариях. Она помогает разработчикам лучше понять логику состояний и переходов (информации, физических ресурсов).

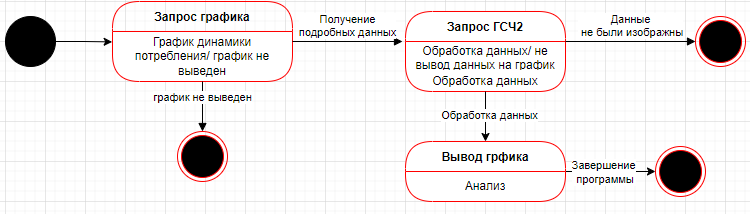


Рисунок 8 – Диаграмма состояний

## 3.2 Структурные диаграммы

На рисунке 9 мы видим диаграмму классов, диаграмма классов представляет описание структуры классов в системе и их взаимосвязи. Она отображает как статические аспекты системы, включая классы, атрибуты и методы, а также динамические аспекты, такие как связи между объектами и выполнение методов во время выполнения программы.

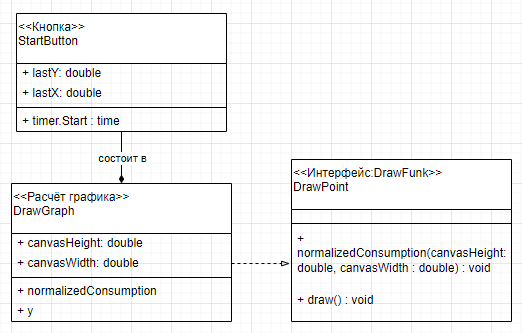


Рисунок 9 – Диаграмма классов

На рисунке 10 мы видим диаграмму компонентов, это одна из структурных диаграмм, которая описывает структуру системы, отражая взаимосвязи между ее компонентами. В отличие от той же диаграммы классов, где мы изучали подробную информацию (входные-выходные данные, связи внутри БД), в диаграмме компонентов мы будем рассматривать более обширный компонент (класс, интерфейс, БД, сервер). Другими словами, мы будем рассматривать обобщенные данные и их связи друг с другом.

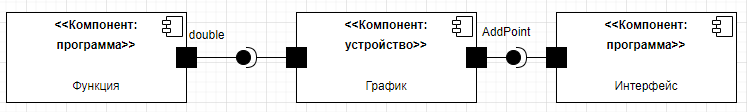


Рисунок 10 – Диаграмма компонентов

На рисунке 11 мы видим диаграмму коопераций, структурная диаграмма, предназначенная для описания поведения системы на уровне самостоятельных объектов. Эти объекты могут обмениваться между собой сообщениями, чтобы достичь поставленной задачи.



Рисунок 11 – Диаграмма коопераций

На рисунке 12 мы видим диаграмму развертывания, – это представление архитектуры исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное (соединяющее) программное обеспечение. Сама по себе аппаратура (аппаратное обеспечение/устройства) без соединительного программного обеспечения была бы просто грудой железа. Поэтому, в данной диаграмме необходимо предусмотреть все пакеты и гаджеты.

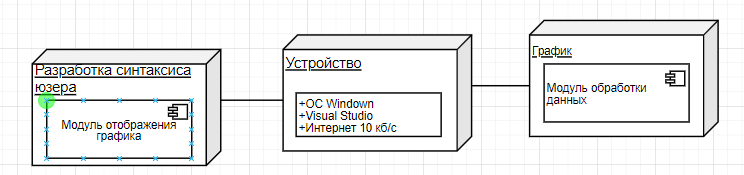


Рисунок 12 – Диаграмма развертывания

## 3.3 Модель С4

Модель С4 (от англ. С4 model, Context-Container-Component-Code model, модель «контекстконтейнер-компонент-код») - достаточно новая и простая архитектурная модель для проектирования системы. Работает по принципу масштабирования картинки. То есть, увеличиваем масштаб - увеличиваем детализацию схемы. Модель предназначена для архитекторов и разработчиков. Также верхний уровень модели подходит для демонстрации заказчикам. Модель описывает систему и её окружающий контекст (пользователи и внешние приложения), состоящую из контейнеров (основные подсистемы), каждый из которых содержит один или несколько компонентов, которые, в свою очередь, реализуются с помощью одного или нескольких кодовых элементов.

На рисунке 13 мы видим 1 уровень (диаграмму контекста), Этот этап «поверхностно» описывает систему для пользователя, которому не нужно вдаваться в технические подробности системы.

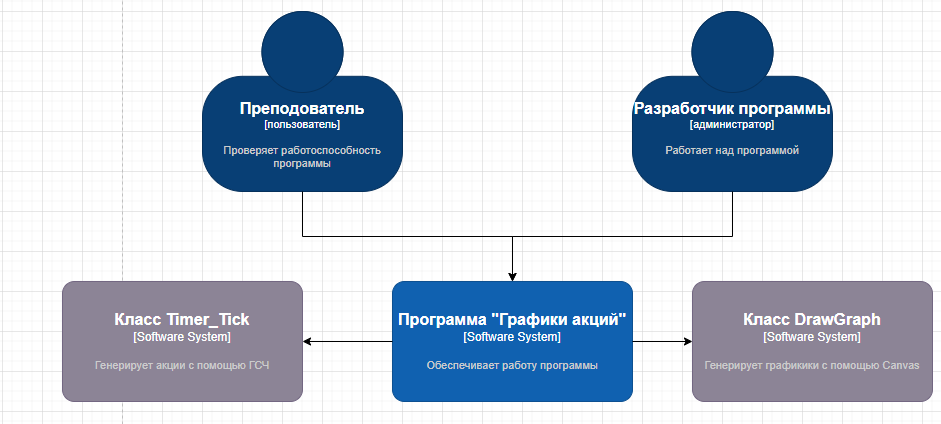


Рисунок 13 – Диаграмма контекста

На рисунке 14 мы видим 2 уровень (диаграмму контейнеров), следующий уровень необходим для описания системы с точки зрения архитектуры приложения. Техническая часть при этом снова не затрагивается. Данная часть наиболее практична для написания документации.

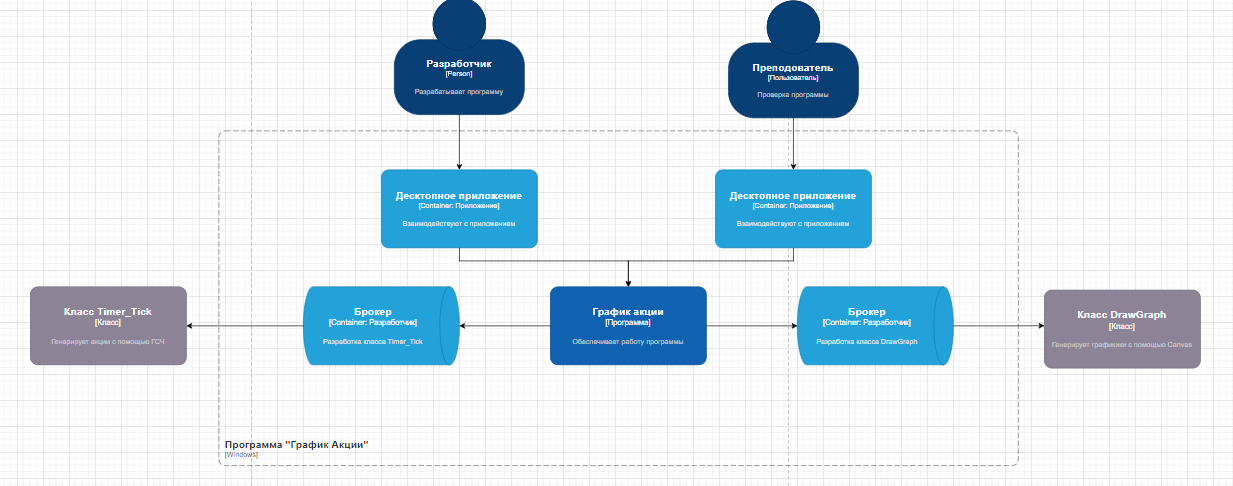


Рисунок 14 – Диаграмма контейнеров

Диаграмма компонентов 3 уровень представлена на рисунке 10, целевая аудитория этого уровня абстракции – программисты и архитекторы. Речь пойдет напрямую о технических характеристиках и рисках.

Диаграмма кода (классов) 4 уровень представлена на рисунке 9, диаграмма кода используется для низкоуровневой детализации системы. На официальной странице модели С4 рекомендуют использовать диаграмму классов, ER-диаграмму или схожие нотации.

# **4 Тестирование ПО**

## 4.1 Тестирование программы

Данная тестовая спецификация описывает тестирование функциональности программы для вывода графика. Программа должна корректно выводить график, выводить на потребления электроэнергии.

Тестирование будет проводиться на следующих аспектах:

* генерация случайных слов;
* вывод графиков;

В данной таблице мы видим тестирование методом черного ящика.

Таблица 5 – Тестирование методом черного ящика

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ теста** | **Описание** | **Действия** | **Ожидаемый результат** | **Фактический результат** |
| 1 | Генерация случайного числа | Нажать на кнопку «Запустить» | Автоматически генерирует случайное число | Положительно |
| 2 | Отображение случайного числа на графике | Нажать на кнопку «Запустить» | Отображение на графике случайных чисел | Положительно |
| 3 | Корректное отображение исходного графика | Нажать на кнопку «Запустить» | Исходный график выводится на экран | Положительно |

Данная тестовая спецификация описывает ключевые тестовые случаи для проверки функциональности программы. Тестирование должно быть проведено на всех перечисленных аспектах, чтобы обеспечить корректную работу приложения.

### **ЗАКЛЮЧЕНИЕ**

В ходе учебной практики было охвачено весь цикл разработки программного обеспечения, включая анализ требований, проектирование, реализацию, тестирование и оценку качества ПО. Практика позволила не только приобрести навыки в разработке и оптимизации алгоритмов, но и углубиться в процессы создания технической документации, таких как разработка диаграмм, написание листинга программы, а также составление протоколов тестирования и анализ метрик программного обеспечения. Большое внимание уделялось обеспечению соответствия программного продукта функциональным требованиям, а также выявлению и устранению ошибок в процессе разработки.

Особое внимание было уделено тестированию программы, с целью выявления дефектов и проверки соответствия спецификациям. В ходе работы также использовались встроенные инструменты для анализа кода и оптимизации программных решений. Итогом работы стала подробная документация, отражающая все этапы разработки, что помогает глубже понять процесс создания качественного программного продукта и его тестирования.

Таким образом, данная учебная практика обеспечила комплексное понимание разработки программного обеспечения и важности качественного проектирования, тестирования и оценки программных решений.

### **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. MicrosoftDocs:C# - документация по языку программирования C#. - [Электронный ресурc] - Режим доступа: https://docs.microsoft.com/en-us/dotnet/csharp/
2. diagrams.net User Guide - руководство по основам работы с draw.io, полезное для начинающих и разработчиков – [Электронный ресурс] – Режим доступа: https://www.diagrams.net/doc/
3. Stack Overflow - один из самых крупных ресурсов для решения проблем, связанных с программированием на C#, WPF –[Электронный ресурс] – Режим доступа: https://stackoverflow.com/
4. MicrosoftDocs:WPF — официальная документация по Windows Presentation Foundation (WPF), которая поможет вам изучить работу с графическим интерфейсом – [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/enus/dotnet/desktop/wpf/
5. Метод черного ящика в тестировании - руководство по тестированию методом черного ящика, которое даст вам понимание о тестирование ПО. – [Электронный ресурс] – Режим доступа: <https://habr.com/ru/>

### **ПРИЛОЖЕНИЕ А**

public partial class MainWindow : Window

{

private Random random = new Random();

private DispatcherTimer timer;

private double lastX = 0;

private double lastY = 0;

public MainWindow()

{

InitializeComponent();

SetupTimer();

}

private void SetupTimer()

{

timer = new DispatcherTimer { Interval = TimeSpan.FromSeconds(1) };

timer.Tick += Timer\_Tick;

}

private void StartButton\_Click(object sender, RoutedEventArgs e)

{

canvas.Children.Clear();

lastX = 0;

lastY = 0;

timer.Start();

}

private void StopButton\_Click(object sender, RoutedEventArgs e)

{

timer.Stop(); // Остановка таймера

}

private void Timer\_Tick(object sender, EventArgs e)

{

if (IsWorkingHours())

{

double consumption = random.Next(20, 100);

DrawGraph(consumption);

}

}

private bool IsWorkingHours() => DateTime.Now.Hour >= 8 && DateTime.Now.Hour < 18;

private void DrawGraph(double consumption)

{

double canvasHeight = canvas.ActualHeight;

double canvasWidth = canvas.ActualWidth;

double normalizedConsumption = (canvasHeight / 100) \* consumption;

double y = canvasHeight - normalizedConsumption;

if (lastX == 0)

{

lastY = y;

lastX += 10;

return;

}

Line line = new Line

{

X1 = lastX - 10,

Y1 = lastY,

X2 = lastX,

Y2 = y,

Stroke = Brushes.Blue,

StrokeThickness = 2

};

canvas.Children.Add(line);

// Добавление текстового блока для отображения значения потребления

TextBlock textBlock = new TextBlock

{

Text = $"{consumption} кВтч",

Foreground = Brushes.Black,

FontSize = 12,

Margin = new Thickness(lastX + 5, y - 15, 0, 0) // Позиционирование текста рядом с точкой

};

canvas.Children.Add(textBlock); // Добавление текстового блока на канвас

lastY = y;

lastX += 10;

if (lastX > canvasWidth)

{

lastX = 0;

canvas.Children.Clear();

}

}

}

}

<Window x:Class="EnergyConsumptionApp.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Динамика потребления электроэнергии" Height="400" Width="600">

<Grid>

<Button Content="Запустить" Click="StartButton\_Click" Width="100" Height="30" HorizontalAlignment="Left" VerticalAlignment="Top"/>

<Button Content="Стоп" Click="StopButton\_Click" Width="100" Height="30" HorizontalAlignment="Right" VerticalAlignment="Top"/>

<Canvas Name="canvas" Margin="10,50,10,10" Background="White"/>

</Grid>

</Window>

### **ПРИЛОЖЕНИЕ Б**

[Blank909909/-2-32 (github.com)](https://github.com/Blank909909/-2-32)