Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

**Технологии разработки программного обеспечения**

**ОТЧЕТ**

ПО РЕЗУЛЬТАТАМ

Производственной практики: технологической



Обучающийся гр. 588-2

Е.В. Кабанова (подпись) (И.О. фамилия)

(дата)

|  |  |
| --- | --- |
|  | Руководитель практики от  профильной организации:  Доцент кафедры КСУП  (должность, ученая степень, звание)  \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ Калентьев А.А.  оценка (подпись) (И.О. Фамилия)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата) |

|  |  |
| --- | --- |
|  | Руководитель практики от  Университета:  Доцент кафедры КСУП  (должность, ученая степень, звание)  \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ Калентьев А.А.  оценка (подпись) (И.О. Фамилия)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (дата) |

Томск 2020

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное

учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра компьютерных систем в управлении и проектировании (КСУП)

УТВЕРЖДАЮ

Зав. кафедрой КСУП

Шурыгин Юрий Алексеевич\_\_\_

(Ф.И.О.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

ЗАДАНИЕ

**на учебную практику: по получению профессиональных умений**

**и опыта профессиональной деятельности**

студенту гр.\_588-2\_ факультета вычислительных систем

Кабанова Екатерина Вадимовна

(Ф.И.О студента)

1. **Тема практики**: разработка программного обеспечения.

2. **Цель практики**: освоение процесса разработки программного обеспечения.

3. **Задачи практики**:

— создать внутреннюю инфраструктуру для разработки ПО;

— написать логику приложения;

— разработать пользовательский интерфейса;

— автоматизировать тестирование;

— собрать установочный пакет;

— представить проект на приёмочном тестировании;

— написать техническую документацию к программе;

— проведение ретроспективы.

4. **Исходные данные для практики**: инструктивные документы в исследуемой области, учебно-методическая литература, интернет-источники.

5. **Технические требования к отчету по практике**: ОС ТУСУР 01–2013.

Дата выдачи: «16» сентября 2020 г.

|  |  |  |
| --- | --- | --- |
| Руководитель практики от университета | |  |
| Доцент кафедры КСУП  (должность) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | | | Калентьев А.А.  (Ф.И.О.) |

Согласовано:

Руководитель практики от профильной организации

|  |  |  |
| --- | --- | --- |
| Доцент кафедры КСУП  (должность) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Калентьев А.А.  (Ф.И.О.) |
|  |  |  |

Задание принял к исполнению «16» сентября 2020 г.



|  |  |  |
| --- | --- | --- |
| Студент гр.\_\_ *588-2*\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | Кабанова.Е.В  *(Ф.И.О.)* |

**Оглавление**

[1 Введение 5](#_Toc55669596)

[2 Ход работы 5](#_Toc55669597)

[2.1 Внутренняя инфраструктура разработки 5](#_Toc55669598)

[2.2 Структура проекта 6](#_Toc55669599)

[2.3 Разработка бизнес-логики 8](#_Toc55669600)

[2.3.1 Класс PhoneNumber 8](#_Toc55669601)

[2.3.2 Класс Contact 9](#_Toc55669602)

[2.3.3 Класс Project 11](#_Toc55669603)

[2.3.4 Класс ProjectManager 13](#_Toc55669604)

[2.3.5 Класс ValueCorrector 13](#_Toc55669605)

[2.3.6 Класс ValueValidator 15](#_Toc55669606)

[2.3.7 История коммитов 18](#_Toc55669607)

[2.4 Разработка пользовательского интерфейса 18](#_Toc55669608)

[2.4.1 Демонстрация пользовательских данных 21](#_Toc55669609)

[2.4.2 Создание нового контакта 22](#_Toc55669610)

[2.4.3 Редактирование контакта 23](#_Toc55669611)

[2.4.4 Удаление контакта 25](#_Toc55669612)

[2.4.5 Поиск контакта 25](#_Toc55669613)

[2.4.6 Панель с уведомлением о дне рождения 26](#_Toc55669614)

[2.4.7 Меню 26](#_Toc55669615)

[2.4.8 Окно About 27](#_Toc55669616)

[2.4.9 История коммитов 27](#_Toc55669617)

[2.5 Юнит-тестирование классов проекта бизнес-логики 28](#_Toc55669618)

[2.5.1 Дерево циклометрической сложности проекта бизнес-логики 28](#_Toc55669619)

[2.5.2 Оценка покрытия тестами 29](#_Toc55669620)

[2.5.3 Пример использования инструментов библиотеки NUnit 30](#_Toc55669621)

[2.5.4 История коммитов 32](#_Toc55669622)

[2.6 Релиз проект 33](#_Toc55669623)

[2.6.1 Создание сценария для сборки установочного пакета и компиляция установщика приложения 33](#_Toc55669624)

[2.6.2 История коммитов 36](#_Toc55669625)

[3 Заключение 37](#_Toc55669626)

[4 Список использованных источников 38](#_Toc55669627)

# **1 Введение**

Цель практики – освоение процесса разработки программного обеспечения, посредствам учебно-методического материала «Теория разработки ПО». Результатом выполнения практики является desktop приложения выполненное согласно назначенному варианту задания. Разработка приложения ведется на языке C# с использованием набора компонентов Windows Forms. В ходе выполнения практики необходимо последовательно выполнить следующие этапы: установить все требуемые приложения, освоить систему версионного контроля Git c использованием сервиса GitHub, написать все необходимые классы логики, разработать пользовательский интерфейс приложения, организовать автоматическое тестирование логики приложения и сборку установщика, а также подготовить техническую документацию в виде пояснительной записки к проекту с UML-диаграммами, ретроспективы по процессу разработки, календарного плана и сметы на разработку аналогичного приложения.

Согласно назначенному варианту в результате выполнения данной работы должно быть разработано и протестировано пользовательское приложение ContactsApp, предназначенное для ведения и хранения контактов.

# **2 Ход работы**

## **2.1 Внутренняя инфраструктура разработки**

Инфраструктура разработки программного обеспечения включает в себя набор программных и аппаратных средств, а также правила взаимодействия разработчиков, обеспечивающих эффективный процесс командной работы. Потребность в четко оговоренной инфраструктуре разработки возрастает с увеличением количества участников проекта. Это объясняется тем, что в большой команде происходит более явное разделение обязанностей. Для обеспечения корректной работы проекта в целом необходимо, чтобы все его отдельные части безупречно функционировали как по отдельности, так и вместе.

В первую очередь необходимо определить набор программных и аппаратных средств, необходимых для разработки приложения. В ходе выполнения данной работы потребовались следующие программы и сервисы:

1. Среда разработки: Microsoft Visual Studio 2018 Community
2. Вспомогательные плагины для среды разработки: JetBrains Resharper.
3. Система версионного контроля: git с использованием сервиса GitHub.com
4. Сборка установочных пакетов: InnoSetup
5. Microsoft Word

Так как данная работа предназначена для индивидуального выполнения, при разработке приложения использовалась упрощенная модель ветвления GitFlow. Основная работа над проектом происходила в ветке разработки – develop. При корректной работе новой функциональности, изменения вносились в главную ветку репозитория – master.

Использование упрощенной модели ветвления GitFlow при работе над приложением ContactsApp показано на рисунке 2.1:

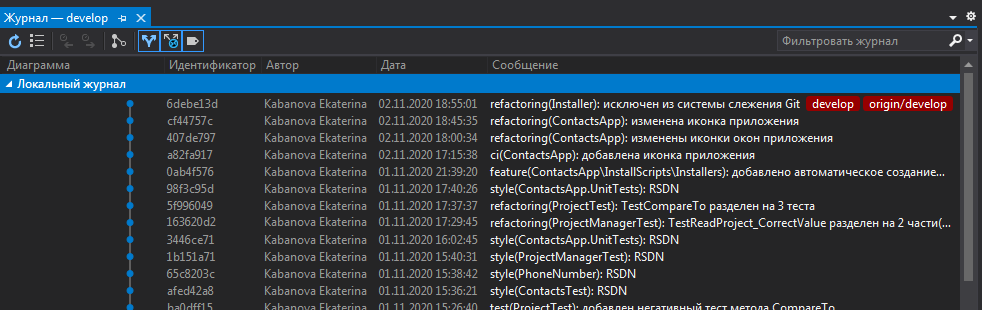


Рисунок 2.1 Использование упрощенной модели ветвления GitFlow при работе над приложением ContactsApp

## **2.2 Структура проекта**

Согласно техническому задания приложение ContactsApp включает в себя 3 проекта:

1. проект логики,
2. проект пользовательского интерфейса,
3. Unit-тесты.

Структура решения ContactsApp приведена на рисунке 2.2:

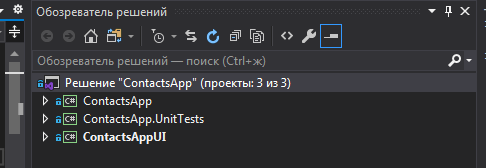


Рисунок 2.2 Структура решения ContactsApp

Сразу после создания проект был размещен на удаленном репозитории сервиса GitHub. Для сокрытия конфигурационных файлов из системы контроля версий Git был задействован файл .gitignore.

Структура удаленного репозитория проекта ContactsApp представлена на рисунке 2.3:

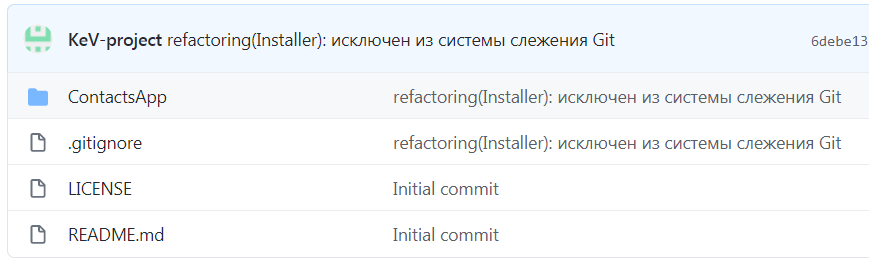


Рисунок 2.3 Структура удаленного репозитория проекта ContactsApp

Структура решения ContactsApp в удаленном репозитории представлена на рисунке 2.4:

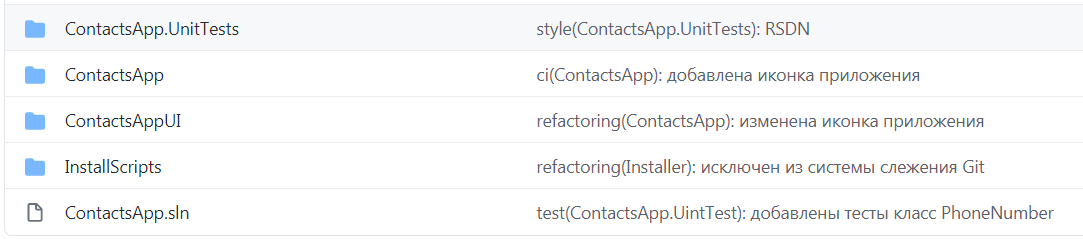


Рисунок 2.4 Структура решения ContactsApp в удаленном репозитории

## **2.3 Разработка бизнес-логики**

Разработка логики приложения – включает написание классов, обеспечивающих хранение и обработку данных, их сохранение и загрузку на жестком диске пользователя. Разрабатываемые классы должны отвечать требованиям оформления кода, документированности, а также защиты целостности данных.

Проект бизнес-логики приложения ContactsApp включает в себя 6 классов: PhoneNumber, Contact, Project, ProjectManager, ValueCorrector, ValueValidator.

UML-диаграмма классов проекта бизнес-логики представлена на рисунке 2.5:

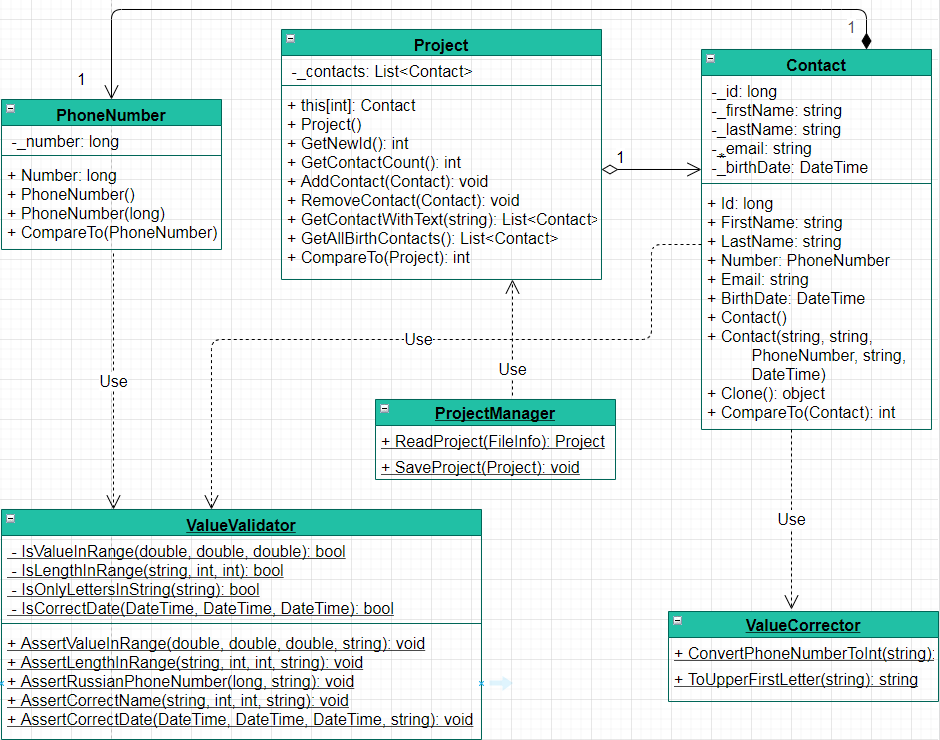


Рисунок 2.5 UML-диаграмма классов проекта бизнес-логики

### **2.3.1 Класс PhoneNumber**

Класс PhoneNumber предназначен для хранения информации о номере телефона контакта. В данной работе класс PhoneNumber содержит только одно закрытое поле – number (номер телефона). Поле number должно быть числовым и содержать ровно 11 цифр. Первая цифра должна быть ‘7’ (только Российские телефонные номера).

Также, класс PhoneNumber содержит конструктор с параметрами, конструктор без параметров и реализацию интерфейса ICombarable, предназначенного для сравнения двух объектов данного класса.

Пример исходного кода, демонстрирующий создание и инициализацию объекта класса PhoneNumber, а также вызов его метода CompareTo представлен на рисунке 2.6:

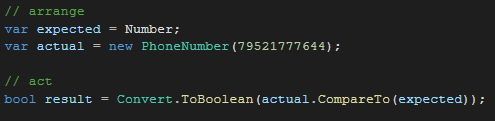


Рисунок 2.6 Пример исходного кода, демонстрирующий создание и инициализацию объекта класса PhoneNumber, а также вызов его метода CompareTo

### **2.3.2 Класс Contact**

Класс Contact предназначен для хранения информации о контакте, а именно: имени, фамилии, номера телефона, даты рождения и адреса электронной почты контакта. В классе организованы соответствующие поля и свойства для каждого из перечисленных параметров: \_firstName (FirstName), \_lastName (LastName), Number, \_birthDate (BirthDate), \_email (Email). Фамилия, имя и e-mail ограничены 50 символами каждое, Id контакта ограничен 15 символами. Первая буква в фамилии и имени должна преобразовываться к верхнему регистру. Дата рождения не может быть более текущей даты и не может быть менее 1900 года. Допустимы контакты с одинаковыми фамилиями и именами.

Также, в классе присутствуют конструктор с параметрами, конструктор без параметров и реализация интерфейсов ICloneable и IComparable, предназначенных для копирования и сравнения объектов данного класса соответственно.

Пример создания и инициализации объектов класса Contact приведен на рисунке 2.7:

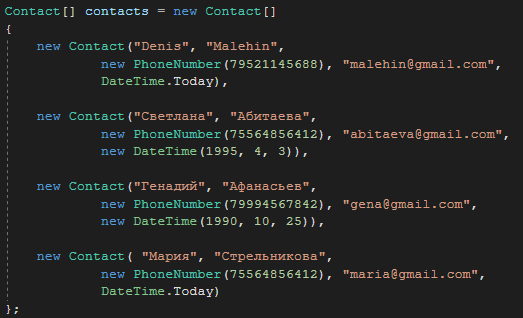


Рисунок 2.7 Пример создания и инициализации объектов класса Contact

Пример вызова метода CompareTo представлен на рисунке 2.8:

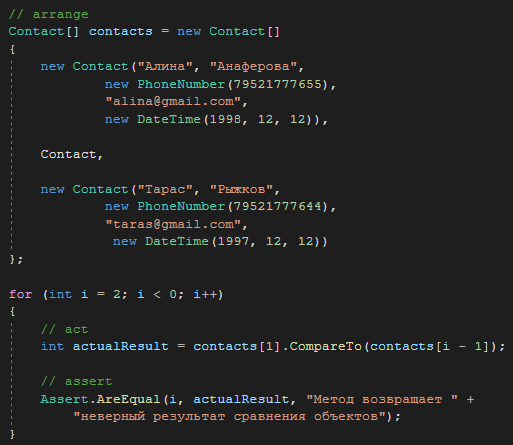


Рисунок 2.8 Пример вызова метода CompareTo

Пример вызова метода Clone представлен на рисунке 2.9:

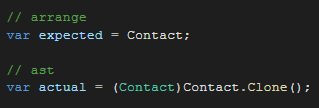


Рисунок 2.9 Пример вызова метода Clone

### **2.3.3 Класс Project**

Класс Project предназначен для хранения пользовательской информации приложения. Все контакты пользователя хранятся в односвязном списке (\_contacts) класса Project. Поле \_contacts – закрытое, поэтому доступ к элементам списка вне класса производится посредствам индексатора.

При успешном добавлении контакта в ContactsApp он попадает в список класса Project, где получает уникальный идентификатор. Для хранения последнего выданного id контакта в классе разработано свойство LastId. При создании объекта класса Project данное свойство инициализируется нулевым значением в конструкторе без параметров. Для получения нового идентификатора используется свойство GetNewId, которое увеличивает значение LastId на единицу и возвращает полученное значение.

Также в классе присутствуют публичные методы, предназначенные для: получения количества элементов списка (GetContactsCount), добавления контакта в список (AddContact), удаления определенного контакта из списка (RemoveContact), получения списка контактов, имя и фамилия которых содержит подстроку (GetContactsWithText), получения списка контактов, у которых сегодня день рождения (GetAllBirthContacts) и реализацию интерфейса IComparable.

Информация, хранящаяся в классе Project, а именно список контактов и последний использованный идентификатор, должны сохранятся после каждого изменения. В данной работе сохранение данных происходит с применением библиотеки Newtonsoft.Json. Для сериализации и десериализации типа ContactsApp.Project был создан контракт данных с применением атрибутов DataContractAttribute и DataMemberAttribute к классу и его членам.

На рисунке 2.10 представлен пример создания и инициализации объекта класса Project, а также вызов его метода AddContact:

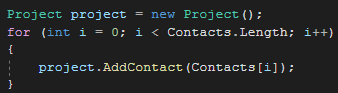


Рисунок 2.10 Пример создания и инициализации объекта класса Project, а также вызов его метода AddContact

На рисунке 2.11 представлен пример вызова метода RemoveContact:

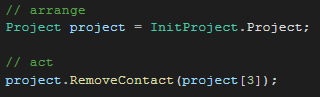


Рисунок 2.11 Пример вызова метода RemoveContact

На рисунке 2.12 представлен пример вызова метода GetContactsWithText:

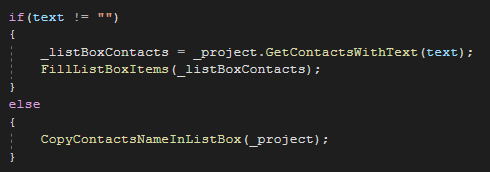


Рисунок 2.12 Пример вызова метода GetContactsWithText

На рисунке 2.13 приведен пример вызова метода GetAllBirthContacts:

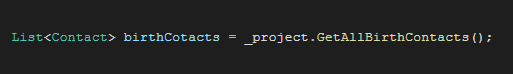


Рисунок 2.13 Пример вызова метода GetAllBirthContacts

На рисунке 2.14 представлен пример вызова метода CompareTo:



Рисунок 2.14 Пример вызова метода CompareTo

### **2.3.4 Класс ProjectManager**

Класс ProjectManager реализует методы SaveProject и ReadProject для сохранения объекта класса Project в файл и загрузки проекта из файла соответственно. Путь к файлу задается при вызове метода в формате FileInfo. Данные в файл записываются в формате json. Для этого используется библиотека Newtonsoft JSON.NET.

Первым делом в методах класса происходит проверка корректности переданного пути к файлу. Если указанная папки или файл не существую, они должны быть созданы автоматически. Если указанный путь к файлу содержит недопустимый формат, возникает исключение. Далее в методах выполняется сериализация или десериализация объекта класса Project.

Пример десериализации объекта приведен на рисунке 2.15:

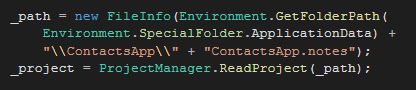


Рисунок 2.15 Пример десериализации объекта

Пример сериализации объекта приведен на рисунке 2.16:



Рисунок 2.16 Пример сериализации объекта

### **2.3.5 Класс ValueCorrector**

Класс ValueCorrector предназначен для корректировки пользовательского ввода. Он включает в себя два статических метода: метод предназначенный для исключения из строки, содержащей номер телефона, всех посторонних символов (ConvertPhoneNumberToInt) и метод для корректировки введенных пользователем имени или фамилии (ToUpperFirstLetter).

Метод ConvertPhoneNumberToInt исключает из строки нечисловые символы, заложенные в маске для введения номера телефона контакта и возвращает число типа long.

Пример вызова метода ConvertPhoneNumberToInt представлен на рисунке 2.17:

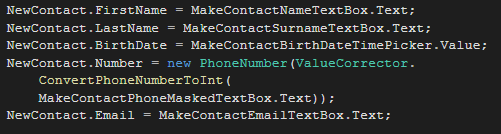


Рисунок 2.17 Пример вызова метода ConvertPhoneNumberToInt

Метод ToUpperFirstLetter приводит первый символ строки к верхнему регистру.

Пример вызова метода ToUpperFirstLetter представлен на рисунке 2.18:

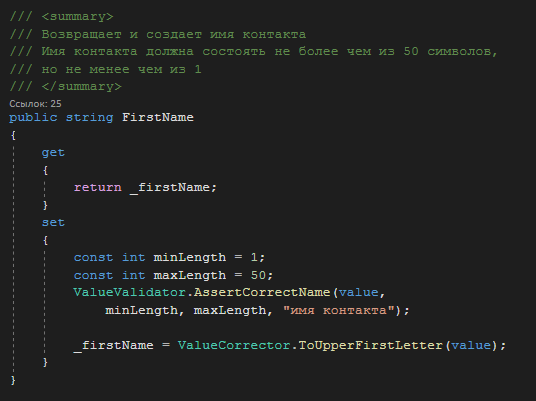


Рисунок 2.18 Пример вызова метода ToUpperFirstLetter

### **2.3.6 Класс ValueValidator**

Класс ValueValidator предназначен для проверки корректности значений перед их использованием. Он содержит пять открытых методов, которые генерируют исключения в случае несоответствия проверяемого значения некоторому набору условий:

Метод AssertValueInRange предназначен для генерации исключения в случае, если число не входит в допустимый диапазон значений.

Пример вызова метода AssertValueInRange приведен на рисунке 2.19:

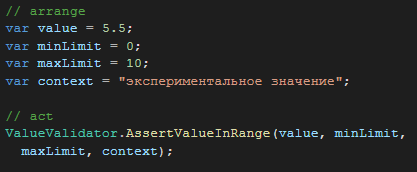


Рисунок 2.19 Пример вызова метода AssertValueInRange

Метод AssertLengthInRange предназначен для генерации исключения в случае, если количество символов в строке не входит в допустимый диапазон.

Пример вызова метода AssertLengthInRange приведен на рисунке 2.20:

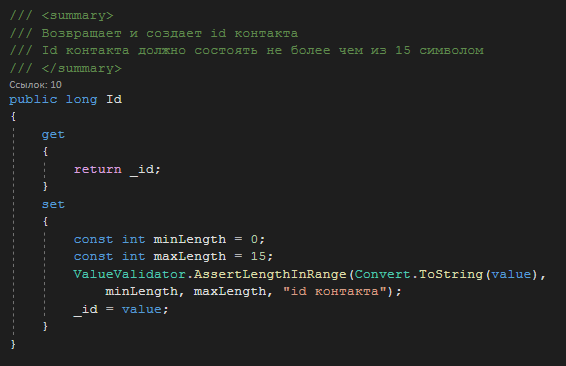


Рисунок 2.20 Пример вызова метода AssertLengthInRange

В перечисленных функциях первым аргументом передается проверяемое значение, далее минимальное, максимальное допустимые значения проверяемого аргумента или минимальное, максимальное значения его длины и строка, содержащую описание члена класса, который будет инициализирован проверяемым значением в случае успешного прохождения проверки.

Метод AssertRussianPhoneNumber предназначен для генерации исключения в случае, если номер телефона не соответствует российским стандартам телефонных номеров.

Вызова метода AssertRussianPhoneNumber представлен на рисунке 2.21:

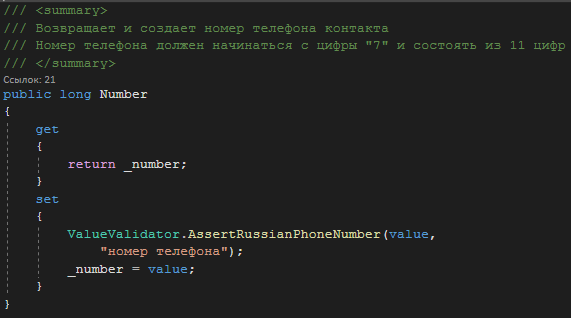


Рисунок 2.21 Пример вызова метода AssertRussianPhoneNumber

Данный метод проверяет, является ли первая цифра проверяемого числа цифрой 7, а также, состоит ли проверяемое число из 11 цифр.

Метод AssertCorrectName предназначен для генерации исключения в случае, если имя и фамилия контакта содержат более 50 символов или содержат символы кроме букв.

Данный метод объединяет в себе 2 проверки. Первая происходит при помощи вызова метода AssertLengthInRange, вторая с использование запроса LINQ.

Пример вызова метода AssertCorrectName представлен на рисунке 2.22:

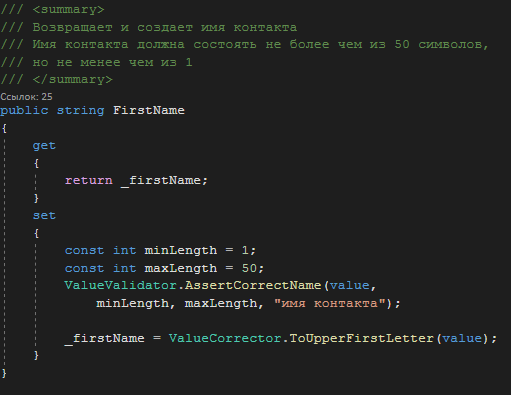


Рисунок 2.22 Пример вызова метода AssertCorrectName

Метод AssertCorrectDate предназначен для генерации исключения в случае, если дата не входит в допустимый временной промежуток.

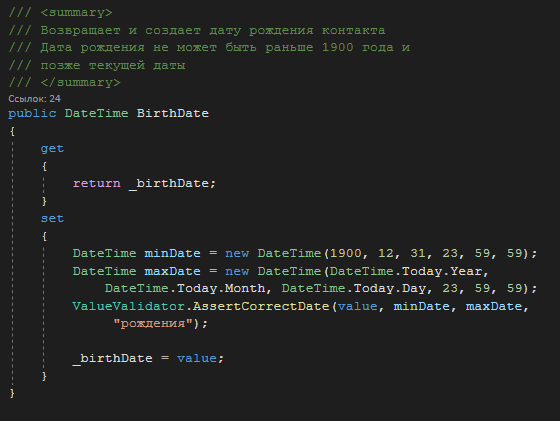
Пример вызова метода AssertCorrectDate представлен на рисунке 2.23:

Рисунок 2.23 Пример вызова метода AssertCorrectDate

Стоит отметить, что класс также содержит 4 закрытых метода, которые используются методами Assert для проверки значений.

### **2.3.7 История коммитов**

На протяжении всего времени разработки приложения активно использовалась система контроля версий Git.

История коммитов получившаяся в результате разработки первоначальной архитектуры классов проекта бизнес-логики представлена на рисунке 2.24:

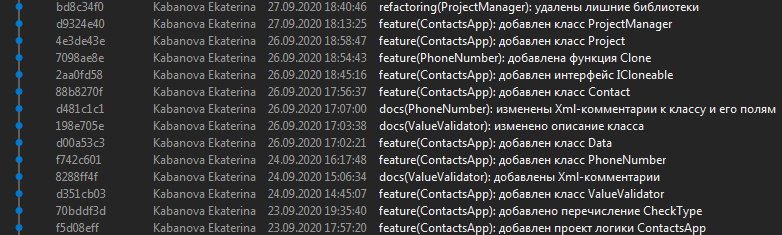


Рисунок 2.24 История коммитов получившаяся на протяжении первоначального этапа разработки классов проекта бизнес-логики

Необходимо отметить, что разработка классов бизнес-логики на данном этапе не закончилась. По мере внесения изменений в другие проекты решения, данный проект непрерывно подвергался рефакторингу.

## **2.4 Разработка пользовательского интерфейса**

Разработка пользовательского интерфейса включает создание компонентов пользовательского интерфейса согласно представленным в техническом задании макетам. Интерфейс должен обеспечивать проверку и защиту от ввода некорректных данных.

В данной работе разработка пользовательского интерфейса предполагает использование компонентов Windows Forms. Формы и элементы управления представляют собой точно такие же классы и объекты, как и любые другие типы данных в ООП, поэтому для разработки пользовательского интерфейса был создан отдельный класс ContactsAppUI. Созданный класс в конечном итоге объединил в себе 3 формы: форму стартового окна, форму для создания и редактирования контакта и форму, содержащую контактную информацию о разработчике.

Работа с приложением начинается со стартового окна, реализованного в главной форме (MainForm). Создание главной формы и запуск стартового окна осуществляется в методе Main класса Program.

Создание и запуск стартового окна представлены на рисунке 2.25:

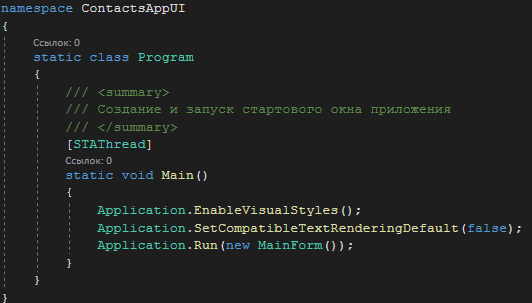


Рисунок 2.25 Создание и запуск стартового окна приложения

При создании формы вызывается конструктор без параметров. В нем реализуется десериализация проекта из файла по указанному пути, а также создание и инициализация компонентов формы. После этого происходит заполнение списка контактов, отображающегося на главном окне приложения. Также при каждом запуске приложения вызывается метод GetAllBirthContacts для получения списка именинников. Из этого списка формируется строка из имен и фамилий именинников для дальнейшего вывода полученной информации на панель главного окна приложения.

Создание и инициализация компонентов любой формы происходит в методе InitializeComponent. Реализация данного метода находится в файле .Designer.cs. При создании интерфейса через конструктор форм данный код генерируется автоматически.

Создание компонентов формы MainForm представлено на рисунке 2.26:

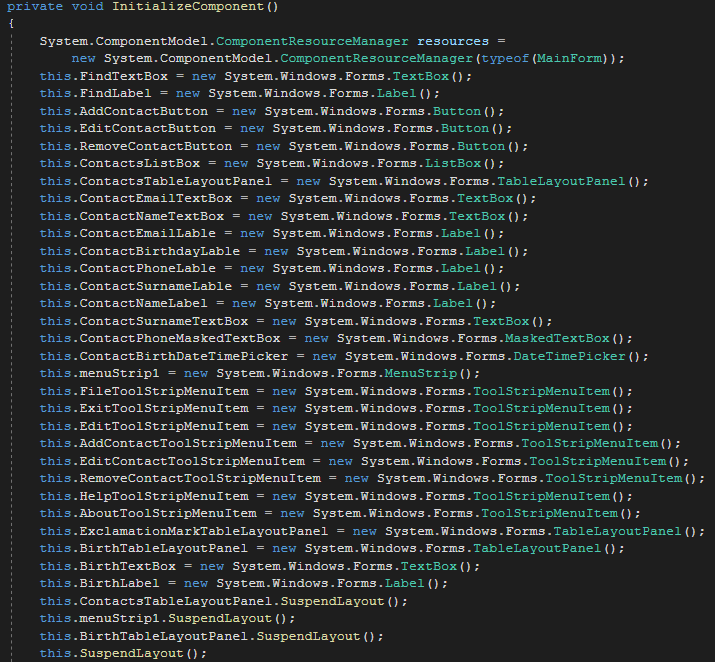


Рисунок 2.26 Создание компонентов формы MainForm

Стартовое окно приложения ContactsApp представлено на рисунке 2.27:

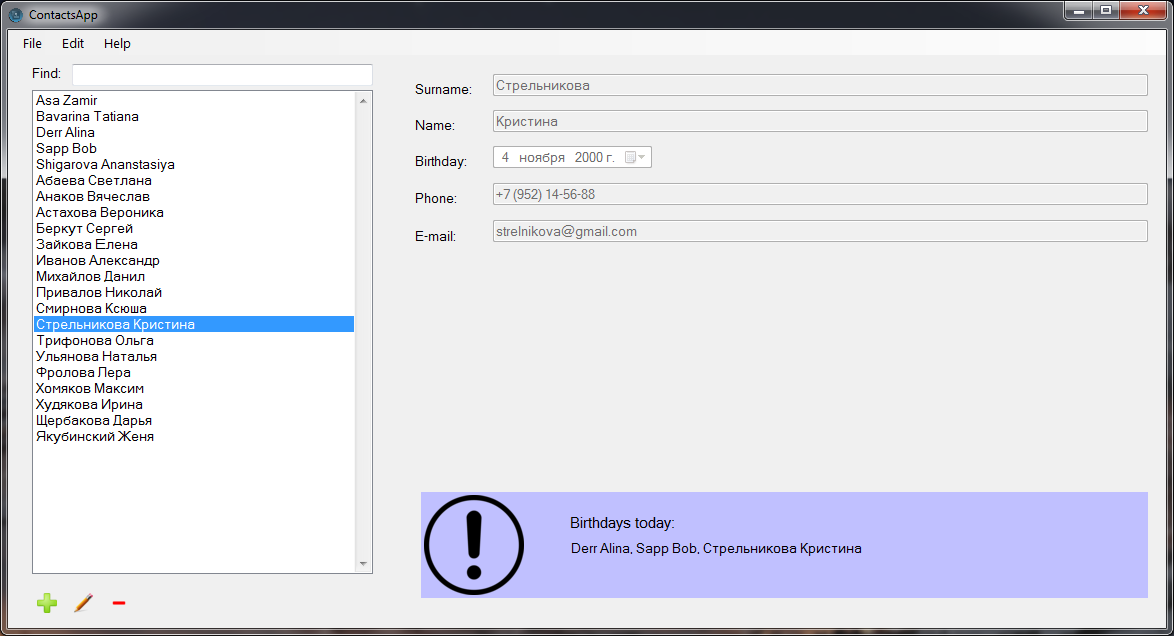


Рисунок 2.27 Стартовое окно приложения ContactsApp

Интерфейс главного окна соответствует принципам адаптивной верстки. Окно имеет фиксированный минимальный размер (1174×636 px).

### **2.4.1 Демонстрация пользовательских данных**

В левой панели главной формы отображается список всех контактов(ContactsListBox) пользователя в алфавитном порядке. При выборе контакта из списка информация о нем отображается в правой панели главного окна (ContactsTableLayoutPanel). Данная функциональность реализована в обработчике событий ContactsListBox\_SelectedIndexChanged. Допустимо редактирование и удаление выбранного контакта.

Элемент ContactsListBox разработан с применением принципов адаптивной верстки. При помощи свойства Anchor элемент привязан к левому краю и растягивается по вертикали при изменении размеров главного окна. При увеличении списка контактов до длины, превышающей текущую вместимость ContactsListBox, в окне появляется прокрутка, позволяющая просмотреть весь список.

Как уже было сказано ранее, информация о выбранном контакте отображается в правой панели главного окна, которая представляет собой экземпляр компонента TableLayoutPanel. Панель содержит в себе 4 элемента типа TextBox:

* ContactSurnameTextBox – выводит информацию о фамилии контакта,
* ContactNameTextBox – выводит информацию о имени контакта,
* ContactPhoneMaskedTextBox – выводит информацию о номере телефона контакта (вывод информации в данном поле контролируется маской),
* ContactEmailTextBox – выводит информацию об адресе электронной почты контакта,

элемент типа DateTimePicker для отображения даты рождения контакта, а также 5 элементов типа Lable для обозначения выводимой информации. Внесение информации во все перечисленные элементы управления не допускается.

### **2.4.2 Создание нового контакта**

В нижнем левом углу окна располагаются 3 элемента типа Button: AddContactButton, EditContactButton и RemoveContactButton, предназначенные для добавления, редактирования и удаления контакта соответственно.

При нажатии на кнопку с иконкой зеленого плюсика, срабатывает обработчик событий AddContactButton\_Click. В нем создается форма ContactForm и запускается окно для создания контакта. При создании формы ContactForm в нее передается объект класса Contact, который в результате должен быть инициализирован с использованием пользовательского ввода.

Окно для создания контакта представлено на рисунке 2.28:

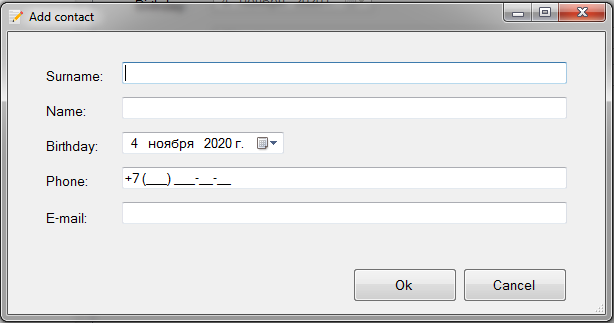


Рисунок 2.28 Окно для создания контакта

Основным элементом данного окна является экземпляр компонента TableLayoutPanel, созданный по аналогии с элементом TableLayoutPanel, расположенным на главной форме. Для добавления нового контакта в список необходимо ввести корректные данные в предложенные поля и нажать кнопку Ok. После этого вызывается обработчик событий MakeContactOkButton\_Click в котором совершается попытка инициализировать переданный объект класса Contact введенными значениями. В случае успешной инициализации окно закрывается и управление возвращается к обработчику событий главной форма AddContactButton\_Click. Здесь происходит добавление нового контакта в список класса Project, изменение содержимого элемента ContactsListBox и сохранение измененного проекта в указанный файл.

В случае несоответствия введенных данных стандартам, заложенным в проекте логики, пользователь получит сообщение с дублированием некорректных данных и причиной исключения. После нажатия на кнопку Ok в данном окне, управление перейдет обратно к форме для создания контакта.

Пример обработки некорректного пользовательского ввода приведен на рисунке 2.29:

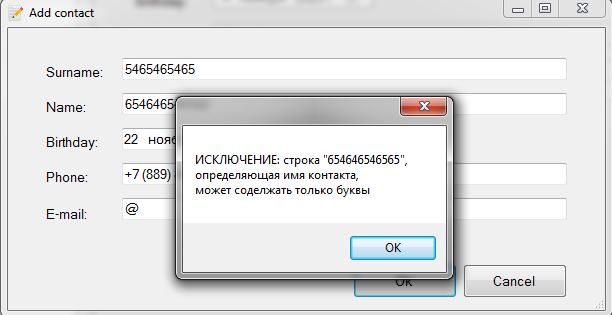


Рисунок 2.29 Пример обработки некорректного пользовательского ввода

### **2.4.3 Редактирование контакта**

Помимо кнопки для создания контакта в нижнем левом углу главной формы располагается элемент EditContactButton, предназначенный для редактирования выбранного контакта.

Для редактирования контакта, нужно выбрать необходимый контакт в списке и нажать на кнопку с иконкой карандаша. При попытке отредактировать заранее не выбранный контакт, пользователь получит следующее предупреждение (рисунок 2.30):

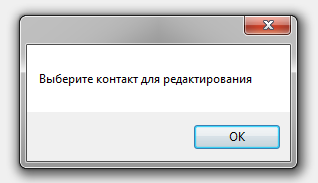


Рисунок 2.30 Сообщение при попытке редактирования неизвестного контакта

После выбора контакта и нажатия на кнопку редактирования срабатывает обработчик событий EditContactButton\_Click. В нем создается форма ContactForm и запускается окно для редактирования контакта. В отличии от создания контакта, при редактировании в форму ContactForm передается объект класса Contact, соответствующий выбранному объекту списка ContactslistBox. Данные переданного контакта вносятся в соответствующие элементы TextBox открывшегося окна.

Окно для редактирования контакта представлено на рисунке 2.31:

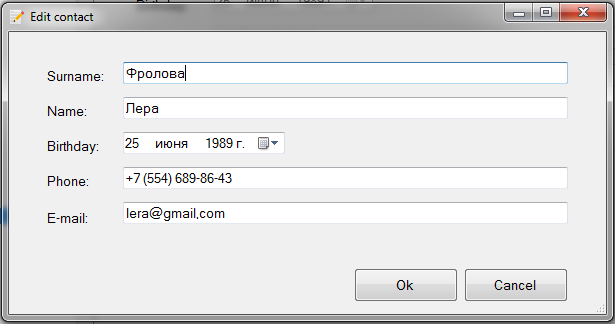


Рисунок 2.31 Окно для редактирования контакта

Обработка некорректно введенных данных в данном окне происходит также, как и в окне для создания контакта.

После корректного исправления контактной информации и нажатия кнопки Ok, управление возвращается в обработчик событий главного окна. Здесь происходит удаление старой версии редактируемого контакта и добавление нового контакта в список класса Project, изменение содержимого элемента ContactsListBox и сохранение обновленного проекта в указанный файл.

При нажатии на кнопку Cancel окно для создания и редактирования закрывается и управление возвращается к главному окну.

Следует отметить, что ContactsForm имеет фиксированный размер (613×320 px).

### **2.4.4 Удаление контакта**

Для удаления контакта нужно выбрать контакт из списка и нажать на соответствующую кнопку. После этого появится окно для подтверждения операции.

Окно для подтверждения удаления контакта показано на рисунке 2.32:

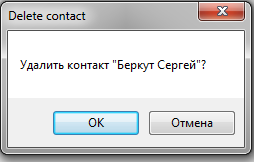


Рисунок 2.32 Окно для подтверждения удаления контакта

### **2.4.5 Поиск контакта**

Над элементом ContactsListBox располагается текстовое поле для поиска контакта по имени или фамилии (FindTextBox). При введении строки в FindTextBox срабатывает обработчик событий FindTextBox\_TextChanged, который считывает введенную строку и вызывает метод GetContactsWithText элемента project, для поиска контакта, имя и фамилия которого содержат текущую подстроку. Вызванный метод возвращает список подходящих контактов, которые помещаются в список элемента ContactsListBox для отображения на главном окне приложения. Далее пользователь может посмотреть информацию о найденных контактах, отредактировать их или удалить.

Пример работы поиска контакта представлен на рисунке 2.33:

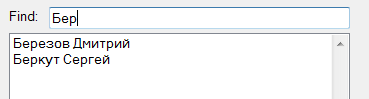


Рисунок 2.33 Пример работы поиска контакта

### **2.4.6 Панель с уведомлением о дне рождения**

Как уже было описано ранее, при каждом запуске приложения в конструкторе главной формы формируется список с контактами, у которых день рождения выпадает на текущий день. Если такие контакты имеются, на главном окне появляется уведомление со списком всех именинников.

Пример уведомления о дне рождении контактов представлен на рисунке 2.34:

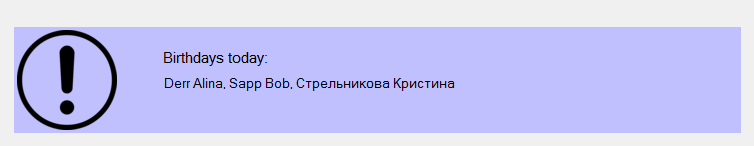


Рисунок 2.34 Пример уведомления о дне рождении контактов

Данная панель представляет собой экземпляр компонента TableLayoutPanel, включающий знак уведомления, метку с текстом “Birthdays today” и элемент типа TextBox, отображающий список именинников. Элемент TextBox не доступен для редактирования.

Панель адаптивна к изменениям родительской формы. Она привязана к нижнему краю и растягивается по горизонтали в результате изменения ширины главного окна. Такое поведение панели достигнуто с помощью свойства Anchor.

### **2.4.7 Меню**

Меню располагается вдоль верхнего края главного окна и представляет собой экземпляр компонента StripMenu. Оно включает в себя вкладки File, Edit и Help. Элемент File имеет дочерний элемент типа StripMenuItem, организующий выход из приложения. Элемент Edit содержит коллекцию элементов типа StripMenuItem, дублирующих функционал создания, редактирования и удаления контакта. Вкладка Help включает в себя элемент About, который вызывает форму с информацией о приложении.

### **2.4.8 Окно About**

При нажатии клавиши F1 или выбрав элемент About в меню главного окна вызывается обработчик событий AboutToolStripMenuItem\_Click, который создает форму AboutForm и открывает окно с информацией о приложении. Данное окно содержит информацию о названии приложении, версию приложения, инициалы разработчика, ссылку на электронную почту и на репозиторий с приложением.

Окно About представлено на рисунке 2.35:

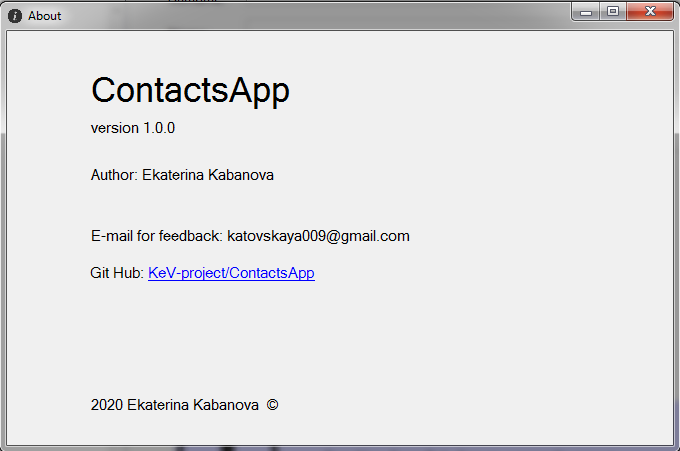


Рисунок 2.35 Окно About

### **2.4.9 История коммитов**

История коммитов получившаяся в результате разработки пользовательского интерфейса представлена на рисунке 2.36:

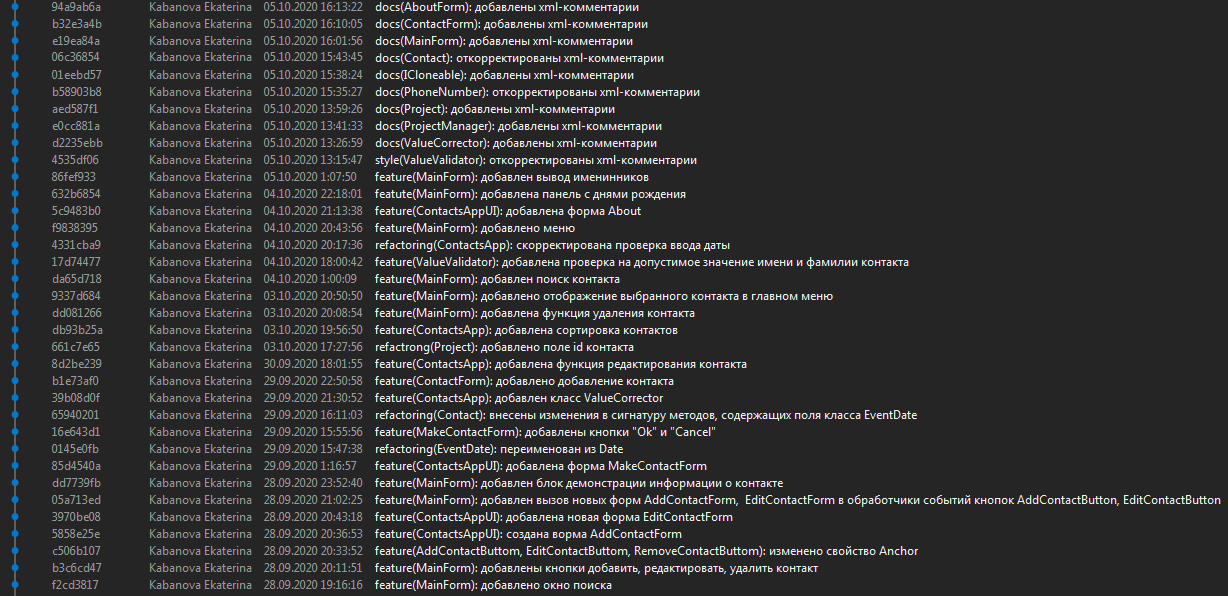


Рисунок 2.36 История коммитов получившаяся в результате разработки пользовательского интерфейса

Необходимо отметить, что разработка классов пользовательского интерфейса на данном этапе не закончилась. По мере внесения изменений в другие проекты решения, данный проект непрерывно подвергался рефакторингу.

## **2.5 Юнит-тестирование классов проекта бизнес-логики**

Тестирование – процесс проверки соответствия объекта заявленным требованиям.

Юнит-тестирование – тестирование минимальных модулей архитектуры, максимально изолированных друг от друга. Такими модулями как правило являются классы. Условие изоляции тестируемого модуля от других необходимо для того, чтобы в случае обнаружения ошибки быть уверенным, что ошибка возникла именно в тестируемом модуле. В противном случае найти место возникновении ошибки будет гораздо сложнее.

### **2.5.1 Дерево циклометрической сложности проекта бизнес-логики**

Чтобы рассчитать циклометрическую сложность проекта, сначала необходимо рассчитать циклометрическую сложность каждого метода, затем просуммировать циклометрическую сложность методов для каждого класса и рассчитать циклометрическую сложность проекта как сумму сложностей всех его классов.

Рассчитаем циклометрическую сложность проекта бизнес-логики:

Проект ContactsApp (цикл. сложность проекта = 66)

1. Класс PhoneNumber (цикл. сложность класса = 7)
2. Класс Contact (цикл. сложность класса = 21)
3. Класс Project (цикл. сложность класса = 14)
4. Класс ProjectManager (цикл. сложность = 9)
5. Класс ValueCorrector (цикл. сложность = 3)
6. Класс ValueValidator (цикл. сложность = 12)

Для сто процентного покрытия тестами класса бизнес-логики потребовались разработать 52 теста.

Общая циклометрическая сложность и время выполнения тестов классов бизнес-логики и проекта в целом представлено на рисунке 2.37:

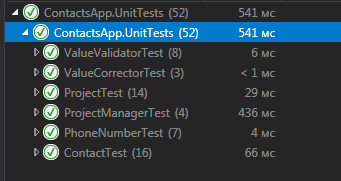


Рисунок 2.37 Общая циклометрическая сложность и время выполнения тестов классов бизнес-логики и проекта в целом

### **2.5.2 Оценка покрытия тестами**

Для проверки достаточности написанных тестов использовалось заранее установленное расширение JetBrains Resharper – платная утилита, устанавливаемая поверх Visual Studio, содержащая полезные инструменты для написания кода. Утилита в реальном времени анализирует написанный код, находит в нём ошибки и предлагает варианты их исправления. Благодаря этому значительно ускоряется процесс написания кода, а также его качество.

Также плагин имеет интеграцию с библиотеками автоматизированного тестирования, позволяя проводить тестирование ПО непосредственно в среде разработки Visual Studio.

Для студентов вузов существует бесплатная лицензия, предоставляемая при отправке фотографии студенческого билета или указании персонального университетского e-mail.

Анализ покрытия проекта тестами, выполненный с использованием Resharper представлен на рисунке 2.38:

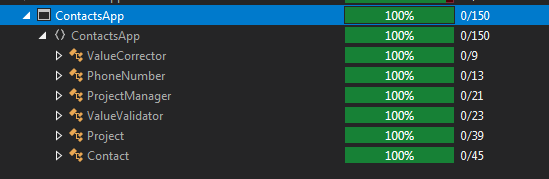


Рисунок 2.38 Анализ покрытия проекта тестами, выполненный с использованием Resharper

### **2.5.3 Пример использования инструментов библиотеки NUnit**

Для написания тестов проекта бизнес-логики использовалась библиотека NUnit. Библиотека NUnit предназначена для написания юнит-тестов приложений для платформы .NET и доступна для подключения в проект как пакет NuGet. Для работы с платформой необходимо подключить два пакета:

1. NUnit – непосредственно фреимворк для автоматизации тестирования;
2. NUnit3TestAdapter – дополнительный пакет, позволяющий запускать юнит-тесты в среде Visual Studio. Без данного пакета запуск юнит-тестов придется выполнять в отдельном консольном приложении.

Работа с библиотекой основана на использовании класса Assert для сравнения ожидаемых и полученных данных, и использовании так называемых атрибутов.

В качестве примера использования библиотеки NUnit на рисунке 2.39 представлен позитивный тест конструктора класса Project:

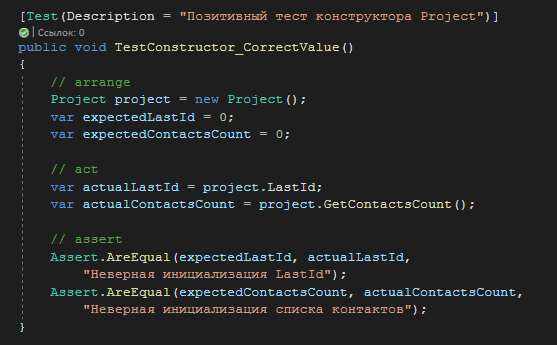


Рисунок 2.39 Позитивный тест конструктора класса Project

В первую очередь, для работы с фреимворком в классе необходимо подключить соответствующее пространство имен (using NUnit.Framework).

Перед методом необходимо указать атрибут [Test]. Внутри атрибута Test можно задать значение для свойства Description и дать описание данного теста. Наличие описания упрощает навигацию в юнит-тестах.

Тест условно разделен на 3 части: arrange – подготовка данных, act – запуск тестируемой функциональности, assert – проверка результата.

В первой части тест, представленного на рисунке 2.39, происходит инициализация тестируемого объекта и определение ожидаемых значений его полей.

В блоке act происходит получение реальных значений созданного объекта.

В третий части при помощи метода AreEqual класса Assert происходит сравнение ожидаемых и действительных значений полей. Если сравниваемые значения не совпадают, тест не пройден, а значит в коде есть ошибка, которая описывается в сообщении, передаваемом в качестве третьего параметра метода.

### **2.5.4 История коммитов**

История коммитов получившаяся в результате разработки юнит - тестов представлена на рисунке 2.40:

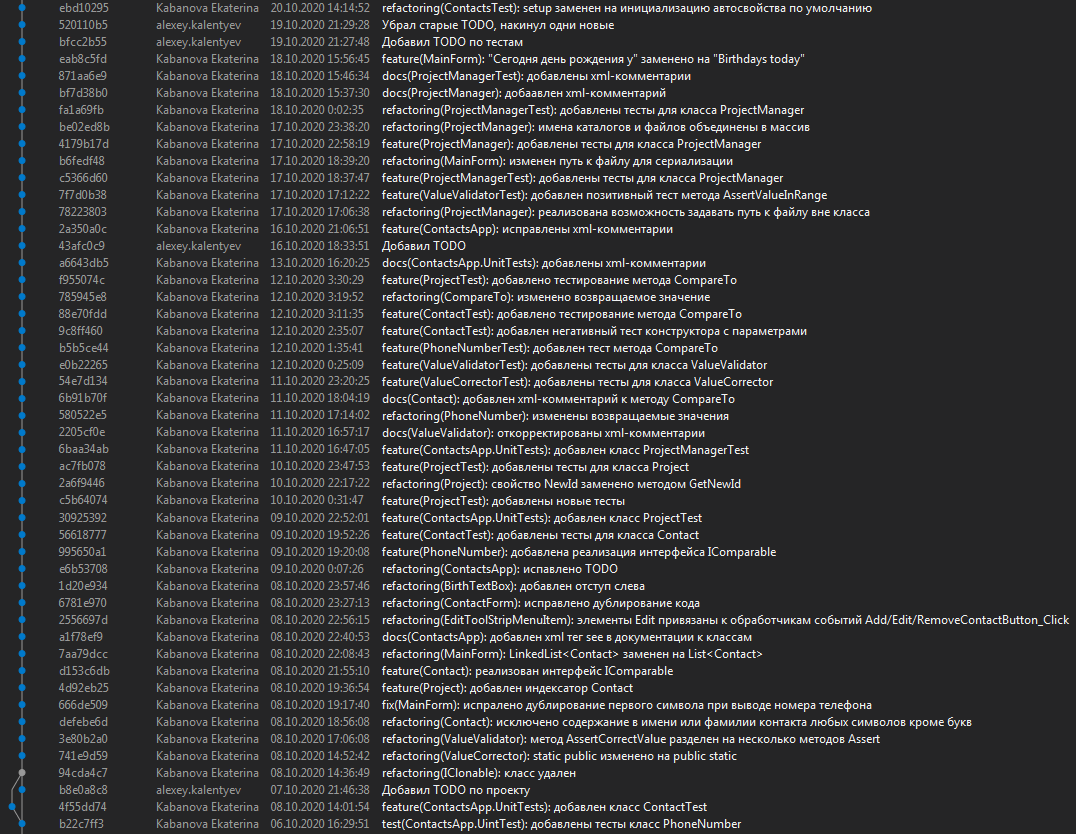


Рисунок 2.40 История коммитов получившаяся в результате разработки юнит-тестов

Необходимо отметить, что разработка юнит-тестов на данном этапе не закончилась. По мере внесения изменений в другие проекты решения, данный проект непрерывно подвергался рефакторингу и доработке.

## **2.6 Релиз проект**

### **2.6.1 Создание сценария для сборки установочного пакета и компиляция установщика приложения**

Прежде всего перед сборкой необходимо выполнить компиляцию решения в конфигурации Release. Таким образом получаем только необходимые для сборки файлы без дополнительного кода для отладки, который будет замедлять работу приложения.

Создание сценария для сборки установочного пакета и компиляция установщика производились автоматически в среде Visual Studio с использованием программы InnoSetup. Для этого в качестве событий после успешной сборки проекта ContactsAppUI были прописаны следующие команды (рисунок 2.41):

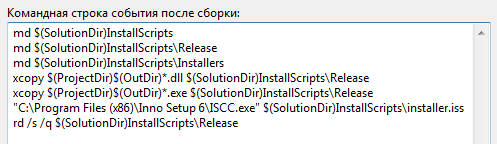


Рисунок 2.41 События после успешной сборки проекта ContactsAppUI

В первую очередь после успешной сборки проекта в папке решения автоматически создается папка InstallScripts, в которой создаются подпапки Release и Installer. В папку Release помещаются файлы необходимые для сборки установочного пакета. Такие файлы имеют расширение \*.exe и \*.dll и представлены на рисунке 2.42:

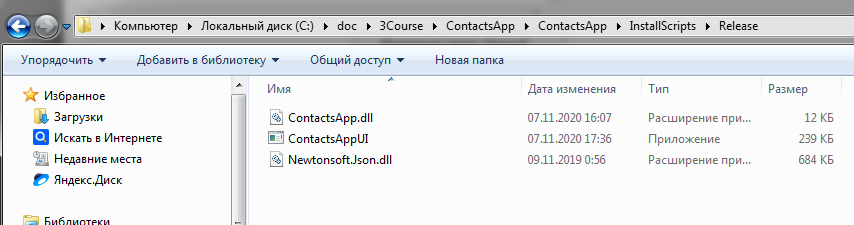


Рисунок 2.42 Содержимое временной папки Release

Данная папка требуется только в процессе сборки, поэтому после создания установщика автоматически удаляется.

Далее происходит создание сценария установочного пакета при помощи программы InnoSetup. Сценарий сохраняется в формате installer.iss и помещается в папку InstallScripts.

Сценарий для сборки установочного пакета приложения ContactsApp представлен на рисунке 2.43:

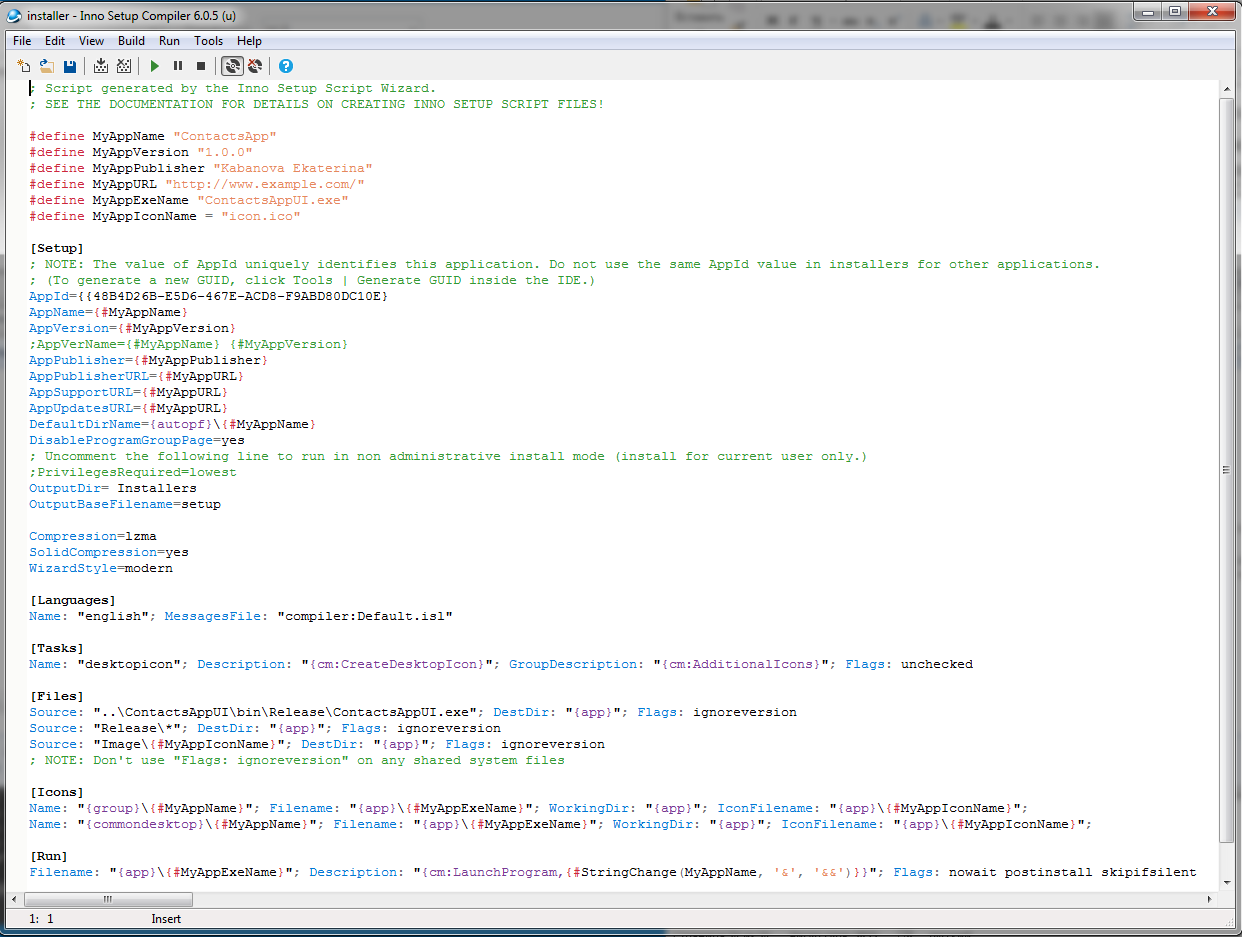


Рисунок 2.43 Сценарий для сборки установочного пакета приложения ContactsApp

Сценарий разделен на область определения переменных, секции с инструкциями, а также может содержать код на языке Delphi для создания дополнительных, нестандартных окон в установщике.

В начале сценария можно объявить различные константы с помощью директивы #define. После директивы #define сначала указывается имя константы и через пробел её значение. В сценарии, представленном на рисунке 2.30, объявлены 6 констант: название приложения, версия выпуска, инициалы разработчика, адрес электронной почты, исполняемый файл проекта ContactsAppUI и имя файла с иконкой для приложения. При дальнейшей поддержке программы, в случае изменения, например, адреса почты достаточно будет исправить только значение переменной вместо исправления значений во всём сценарии.

Секция [Setup] задаёт основные настройки установщика:

* AppId – уникальный идентификационный номер приложения для регистрации в реестре Windows, сгенерированный автоматически,
* AppName – название приложения, в том числе и для регистрации в реестре,
* AppVersion – версия приложения,
* AppPublisher – инициалы разработчика,
* DefaultDirName – путь для создания папки приложения на пользовательском устройстве,
* OutputDir – папка, для сохранения установщика,
* OutputBaseFilename – имя установочного файла,
* Compression – алгоритм сжатия (архивирования) файлов, помещаемых в установщик. OutputBaseFilename – имя выходного файла – файла установщика и др.

Секция [Tasks] позволяет создавать дополнительные окна в установщике с опциями, которые сможет установить пользователь. В сценарии, представленном на рисунке 2.30, такой опцией является установка ярлыка приложения на рабочем столе устройства.

Секция [Files] описывает файлы, которые необходимо поместить в установщик. В данном случае это исполнительный файл проекта ContactsAppUI, файлы временной папки Release и изображение, используемое в качестве иконки приложения.

Далее описывается установка иконки приложения.

Секция [Run] является необязательной. В этой секции можно указать любое количество программ для выполнения после того, как программа была успешно установлена.

### **2.6.2 История коммитов**

История коммитов полученная в результате добавления в проект пользовательского интерфейса автоматической сборки установочного пакета и компиляции установщика представлена на рисунке 2.44:

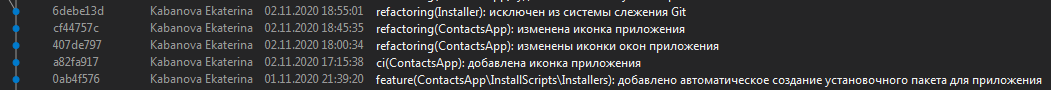


Рисунок 2.44 История коммитов полученная в результате добавления в проект пользовательского интерфейса автоматической сборки установочного пакета и компиляции установщика

# **3 Заключение**

В ходе выполнения данной работы были изучены основные принципы разработки программного обеспечения. Все полученные знания были закреплены на практике в процессе разработки десктоп-приложения для ведения и обработки контактов ContactsApp.

В процессе подготовки к созданию приложения были изучены основные принципы разработки на языке C# и набор компонентов Windows Forms.

Согласно техническому заданию были разработаны проекты бизнес-логики и пользовательского интерфейса. При помощи библиотеки NUnit были разработаны и проведены тесты для всех классов бизнес-логики.

После завершения разработки всех частей приложения при помощи программы InnoSetup был создан сценарий установочного пакета и скомпилирован установщик приложения.

На протяжении всего процесса разработка результаты публиковались в удаленном репозитории на сайте GitHub.

# **4 Список использованных источников**

1. Технологии разработки программного обеспечения: методические указания к самостоятельной и лабораторным работам / А. Е. Горяинов, А.А. Калентьев, 2014. – 176 с.
2. <https://metanit.com/sharp/tutorial/>
3. <https://metanit.com/sharp/windowsforms/>
4. <https://githowto.com/ru>
5. <https://rsdn.org/article/mag/200401/codestyle.XML>
6. <https://ru.stackoverflow.com/>